Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Peter Knut

Editor pro 3D počítačovou hru

Katedra softwarového inženýrství
Vedoucí diplomové práce: *Doc. Ing. Jiří Žára, CSc.*
Studijní program: *Informatika, Softwarové systémy*

Na tomto mieste by som rád poďakoval Doc. Ing. Jiřímu Žárovi, CSc. za sprostredkovanie tejto zaujímavej diplomovej práce a Ing. Jiřímu Rohlíkovi za cenné rady a trpezlivosť počas celej implementácie aplikácie.

Vyhlasujem, že som svoju diplomovú prácu napísal samostatne a výhradne s použitím citovaných prameňov. Súhlasím so zapožičiavaním práce.

V Prahe dňa 1. 8. 2006

Peter Knut

Obsah

Zoznam obrázkov

Název práce: *Editor pro 3D počítačovou hru*

Autor: *Peter Knut*

Katedra: *Katedra softwarového inženýrství*

Vedoucí diplomové práce: *Doc. Ing. Jiří Žára, CSc.*

e-mail vedoucího: *zara@fel.cvut.cz*

Abstrakt: *Každá počítačová hra většího rozsahu používá pro vytvoření svého obsahu specializovaný editor. Toto platí především pro 3D hry. Editor umožňuje nejen sestavovat výsledné scény z herních objektů, ale i nastavovat těmto objektům různé vlastnosti důležité pro daný typ hry. Objekty mohou např. podléhat fyzikální simulaci, být důležité z pohledu umělé inteligence, mít ve virtuálním světě konkrétní roli a pod.*

Tato práce se zabývá vytvořením editoru objektů a scén pro 3D hru založenou na knihovně Nebula Device. Pomocí konfiguračních souborů je aplikace schopná vytvořit jednoduché grafické uživatelské rozhraní propojené se skriptovacími editačními funkcemi v jazyku Lua. Pro uložení konfigurace editoru byla použita technologie XML, vytvoření uživatelského rozhraní zabezpečuje knihovna MFC.

Aplikace zobrazuje i jinak neviditelné herní objekty (kamery a zdroje světla), umožňuje označit a manipulovat s objekty pomocí myši přímo v scéně. Podporuje tři pracovní módy: editaci scény, editaci prototypu a herní mód.

Klíčová slova: *editor hry, Nebula Device, Lua, MFC, OpenDE*

Title: *Editor for 3D computer game*

Author: *Peter Knut*

Department: *Department of software engineering*

Supervisor: *Doc. Ing. Jiří Žára, CSc.*

Supervisor's e-mail address: *zara@fel.cvut.cz*

Abstract: *Every larger computer game uses special editor for creating its sources, especially 3D games. The whole scenes from the game objects are allowed to assemble from the editor, together with all properties which are important for the given type of the game. For example, objects can be affected by physical simulation, or they can be important from the artificial intelligence point of view, or they can have a particular role in virtual world etc.*

The object of this diploma is the implementation of the editor for 3D game based on the Nebula Device library. The application is able to create simple graphic user interface by configuration files. This interface is connected to script functions written in Lua language. XML technology is used for storing the configuration of the editor. Creating the graphic user interface is provided by the MFC library.

The application also allows to show the invisible game objects (cameras and sources of light), to select and manipulate with game objects by the mouse in the scene directly. It supports three working modes: scene editing, prototype editing and play mode.

Keywords: *game editor, Nebula Device, Lua, MFC, OpenDE*

1 Úvod

1.1 Cieľ práce

Každá 3D počítačová hra sa skladá z priestorových objektov, ktoré spolu vytvárajú komplexnú scénu. Objekty majú základné vlastnosti ako sú model (sieť trojuholníkov vytvárajúca povrch objektu), textúra (obrázok, ktorý pokryje model a tým ho vyfarbí), poloha a otočenie. Tieto vlastnosti nám stačia pre zobrazenie výslednej scény.

Na vytvorenie objektov a jednoduchej scény si vystačíme s niektorým z univerzálnych modelovacích nástrojov. V počítačových hrách majú ale jednotlivé objekty svoj ďalší špecifický význam a vlastnosti. Objekty môžu napr. podliehať fyzikálnej simulácii, byť dôležité z pohľadu umelej inteligencie, mať vo virtuálnom svete konkrétnu rolu, prípadne môžu byť veľmi špecializované (ako sú kamera, explózia, zdroj svetla a pod.). Tu už vzniká potreba nástroja – editora, šitého na mieru konkrétnej hre. Samozrejme je veľmi vhodné aby takáto aplikácia bola podľa možností rozšíriteľná a použiteľná pre rôzne hry stavané na rovnakom základe. Editor dokáže pracovať jak s celou scénou (levelom) hry, tak so samostatnými objektmi. Je možné vytvárať nové scény a objekty, otvárať už existujúce a nastavovať im potrebné vlastnosti.

Editor využíva všetky prostriedky danej hry, ku ktorej pridáva editačné funkcie. Nejedná sa teda o úplne nezávislú aplikáciu. Jeho vývoj, funkčnosť a možnosti idú ruka v ruke spolu s vývojom hry. Navyše je v dnešnej dobe pre vytvorenie kvalitnej počítačovej hry nereálne programovať úplne všetko od základu, takže hra a teda i editor využívajú viacero dostupných knižníc a technológií.

Cieľom tejto práce je vytvoriť editor objektov a scén pre 3D hru Mutant podľa zadania firmy Napoleon Games. Nasledujúce kapitoly obsahujú základné prvky tohto zadania spolu s doplnkami, ktoré vznikli postupne vzhľadom k prebiehajúcemu vývoju hry. Plná verzia pôvodného zadania sa nachádza v prílohe A.

Ďalšie kapitoly popisujú použité technológie, predovšetkým knižnicu Nebula Device, ktorá je základom hry Mutant. Nasleduje rozbor riešenia s porovnaním možných prístupov, voľba riešenia a jeho implementácia. Pred záverom sa nachádza ešte ukážka praktického využitia.

1.2 Zadanie práce

- Úloha bude riešená v existujúcom prostredí knižnice Nebula Device s plným vyžitím herného enginu Mutant.

- Editor je schopný vytvoriť používateľské rozhranie podľa popisu v XML alebo skriptu. Toto rozhranie je možné previazať s funkciami napísanými v skripte. Ovládacie prvky dialógov buď volajú skriptové funkcie alebo nastavujú konkrétnou vlastnosť C++ objektu. Tieto vlastnosti môžu byť rôzneho typu (napr. reálne číslo, cesta a meno súboru, trojzložkový vektor, tabuľka a pod.) a preto ku každému typu existuje špecializovaný ovládací prvok.

- Grafické rozhranie obsahuje menu a tri základné dialógy:
    - Zoznam objektov v otvorenej scéne – objekty sú vizuálne odlíšené, je možné ich vyhľadávať a filtrovať podľa typu.
    - Stromový prehľad otvoreného prototypu – jednotlivé typy parametrov sú vizuálne odlíšené. Obsahuje tlačidlá pre manipuláciu s parametrami.
    - Dialóg pre zobrazovanie a nastavovanie vlastností – v tomto dialógu sa zobrazujú ovládacie prvky definované v šablóne označeného herného objektu alebo parametrov.
- Editor pracuje v troch módoch: editácia prototypu, editácia scény (levelu hry) a testovanie scény – tzv. herný mód.
- Editačné funkcie sú pokiaľ možno napísané v skripte, v ostatných prípadoch v C++ ako pevná súčasť aplikácie.
- Pri editácii prototypu je možné pridávať a odoberať požadované parametre, nastavovať im všetky dostupné vlastnosti a vytvárať kontajnery. Niektoré typy parametrov (ako napr. kolízna obálka) sú vizualizované.
- Pri editácii scény je možné pridávať a mazať herné objekty, označiť ľubovoľný objekt alebo skupinu a nastavovať im všetky dostupné vlastnosti. Poloha a otočenie objektov sa dá upravovať aj priamo pomocou klávesových skratiek a pohybu myši. Niektoré neviditeľné objekty (ako napr. kamera alebo zdroj svetla) sú vizualizované.

Vysvetlenie pojmov prototyp, parametre, kontajner a herný objekt sa nachádza v kapitole 2.2.1.

2 Použité technológie

Prostredie pre vývoj aplikácie je značne široké a vyžaduje dobré pochopenie. Celý herný engine stojí predovšetkým na knižnici Nebula Device. Preto popisu jej základných princípov a funkčnosti je venovaná zvýšená pozornosť.

Pre vytvorenie používateľského rozhrania bola použitá štandartná knižnica MFC (Microsoft Foundation Class). Rozhranie je definované pomocou jazyka XML. Funkčnosť rozhrania zabezpečuje z veľkej časti súbor skriptov napísaných v jazyku Lua. Nakoniec je predstavená ešte knižnica OpenDE (Open Dynamic Engine, ODE), ktorú editor využíva pre zisťovanie objektu, na ktorý klepol používateľ tlačidlom myši priamo v zobrazenej scéne.

2.1 Nebula Device

Nebula Device[1] je voľne šíriteľný 3D vizualizačný/herný engine vyvíjaný firmou Radon Labs[2]. Engine je určený pre zobrazovanie scén v reálnom čase, prácu so vstupnými zariadeniami a zdrojovými dátami, prehrávanie zvuku, sieťovú komunikáciu a pod. Je napísaný v jazyku C++, skriptovateľný s využitím jazykov TCL/Tk, Lua, Python a Ruby. Využíva najnovšie technológie v oblasti zobrazovania vrátane plného nasadenia shaderov. Podporuje systém Windows spolu s grafickým rozhraním DirectX 9. Možné je i rozšírenie pre Linux a MacOS X spolu s rozhraním OpenGL.

Najdôležitejšími vlastnosťami enginu sú:

- Hierarchický menový priestor
- Objektová perzistentnosť
- Skriptovanie
- Grafika založená na shaderoch
- Animácie
- Grafické používateľské rozhranie
- Manažment zdrojov
- Audio systém
- Práca so vstupnými zariadeniami

Podrobné informácie o knižnici Nebula Device sa nachádzajú v [6].

2.1.1 Objektová hierarchia

Medzi najvýznamnejšie črty tejto knižnice patrí objektová hierarchia nazývaná *Named Object Hierarchy* (NOH). Jednotlivé C++ objekty majú svoje textové meno a sú usporiadané v stromovej hierarchii pripomínajúcej systém súborov s názvami ciest ako napr. „/sys/servers/gfx“. Meno objektu môže byť kedykoľvek prevedené na C++ ukazateľ a naopak. Takéto označenie a usporiadanie je výhodné napríklad pre prístup k objektom zo skriptu, pri dynamických operáciách, kde nie je vhodné udržiavať priamo ukazateľ na objekt alebo pri ukladaní časti hierarchie objektov do súboru.

[1] http://nebuladevice.cubik.org
[2] http://www.radonlabs.de

Objekty patriace do NOH sa nazývajú Nebula objektmi. Ich C++ triedy majú spoločného predka nazvaného nRoot. Ten zabezpečuje okrem práce so samotnou hierarchiou aj univerzálne skriptové rozhranie, pričom je na výber, či trieda odvodená od nRoot bude obsahovať funkcie skriptového rozhrania alebo nie.

Skriptovanie v Nebula Device znamená volanie skriptových príkazov priamo na C++ objekty. Komunikačným bodom medzi príkazmi skriptovacieho jazyka a objektmi odvodenými od triedy nRoot je skriptový server (nScriptServer a jeho potomkovia pre konkrétny jazyk), ktorý interpretuje skript a prekladá volania funkcií spolu s ich parametrami.

Ďalšou dôležitou vlastnosťou Nebula objektov je ich perzistentnosť. Perzistentnosť objektu znamená, že sa objekt dokáže pomocou skriptového servera automaticky serializovať a uložiť do súboru vrátane všetkých objektov, ktoré sa v NOH nachádzajú pod ním. Spätne je potom možné takýto objekt (či objekty) načítať a obnoviť v pamäti. Vygenerovaný súbor má podobu skriptu, v ktorom sú zapísané príkazy na vytvorenie objektov spolu s ďalšími skriptovými funkciami. Proces serializácie sa dá pre každú triedu predefinovať a zahrnúť do neho všetko, čo potrebujeme. K tomuto účelu sa využívajú práve funkcie skriptového rozhrania objektov. Použitý jazyk pri zapisovaní skriptu závisí od nastaveného skriptového servera. Na výber máme i zjednodušený binárny jazyk, ktorý umožňuje zvýšiť rýchlosť čítania zo súboru. Pri načítavaní sa jazyk zistí automaticky z hlavičky súboru.

Nasleduje ukážka súboru s uloženými objektmi v jazyku Lua.

```
-- ------------------------------------------------------------
-- $parser:nluaserver$ $class:texturecontainer$
-- ------------------------------------------------------------
new('nroot','textury')
  sel('textury')
  new('ntexture','stena')
    sel('stena')
    call('setfilename', [[textury:mesto/stena.n2]])
  sel('..')
  new('ntexture','okno')
    sel('okno')
    call('setfilename', [[textury:mesto/okno.n2]])
  sel('..')
sel('..')
call('setactivetexture', [[okno]])
-- ------------------------------------------------------------
-- Eof
```

V knižnici Nebula Device existuje špecifická skupina objektov, ktoré sa označujú ako servery. Každý server poskytuje ostatným objektom služby z nejakej oblasti. Je tu napríklad zvukový server pre prehrávanie zvukových efektov a hudby; skriptový server pre interpretovanie skriptov niektorého jazyka alebo server pracujúci so vstupnými zariadeniami. Väčšina z týchto serverov môže mať iba jednu inštanciu. V NOH sa nachádzajú spoločne na jednom mieste (/sys/servers).

2.1.2 Grafický subsystém

Grafické zobrazovanie je v Nebule založené na shaderoch. To znamená, že každý stav vykresľovania je riadený shaderovým programom. To umožňuje zobrazovanie a zmenu mnohých efektov bez zásahu do C++ kódu. Shaderové parametre môžu byť vytvorené a menené priamo zo vstavanej príkazovej konzoly.

Hlavnými pojmami v grafickom subsystéme sú *sieť* (*mesh*), *textúra* a *shader*.

Sieť (trieda nMesh2) v sebe zahŕňa vrcholový a indexový zásobník (vertex a index buffer). Vrcholy môžu okrem pozície obsahovať aj ďalšie zložky: tangenty a binomály pre počítanie osvetlenia, hmotnosti a kĺbové indexy pre tzv. vertex skinning apod. Je možné nastaviť pomocné parametre určujúce použitie siete. Siete sú načítavané z textových (.n3d2) alebo binárnych (.nvx2) súborov.

Odporúčaným formátom pre textúry (trieda nTexture2) je DDS. Sú podporované aj iné grafické formáty ale tie sú vhodné iba na testovanie. Spravidla je ich načítavanie pomalšie a môže dôjsť i k strate výkonu vykresľovania. DDS súbory podporujú okrem iného aj kubické a objemové textúry, tiež môžu obsahovať všetky mip-map úrovne.

Shader objekty (nShader2) sa dokážu samé načítať z dátového súboru. Obsahujú všetko potrebné pre vykreslenie siete a textúry vrátane vrcholových a bodových shaderových programov. DirectX implementácia používa súbory s koncovkou .fx (Direct3D Effect File). To, ako je shader vykreslený je čisto záležitosťou gfx serveru.

Fabrikou pre siete, textúry a shadere je gfx server (trieda nGfxServer2). Tá sa stará tiež o otvorenie výstupného okna s hardwarovou podporou 3D grafiky, nastavenie kamery, zobrazovanie základných útvarov, písma a tiež pracuje s kurzorom myši. Tu je treba zdôrazniť, že Nebula nemusí nutne vytvárať vlastné systémové okno. Umožňuje programátorovi zadať identifikátor už vytvoreného okna. V takomto prípade sa vytvorí iba jednoduché dcérske okno bez okrajov v klientskej oblasti zadaného okna.

Zobrazované objekty majú stromovú štruktúru, ktorej uzlami sú potomkovia triedy nSceneNode. Tí poskytujú informácie o transformácii, geometrii, objeme, shaderi, materiále, svetle, tieni a animácii uzla. Výsledná scéna sa stavia a vykresľuje pomocou serveru triedy nSceneServer. Stavanie scény typicky prebieha odznova pred každým vykreslením, pričom dochádza k rôznym optimalizáciám scény. Nedochádza však k žiadnemu automatickému orezávaniu scény na základe viditeľnosti objektov. Preto je potrebné objekty najprv samostatne predspracovať.

Zobraziteľné uzly, odvodené od nSceneNode, môžu byť pripojené do výslednej scény aj niekoľko krát. Viaceré z nich však vyžadujú udržiavanie samostatných informácií medzi jednotlivými vykresleniami; a to pre každú inštanciu zvlášť. Tieto dáta sú zvyčajne uložené v nejakom serverovom objekte. Pre identifikáciu, ktoré dáta patria danej inštancii uzla, slúžia tzv. vykresľovacie kontexty (nRenderContext).

Napríklad máme hierarchiu uzlov, ktorá reprezentuje animovanú postavu a potrebujeme vykresliť skupinu takýchto postáv. Do scény bude teda hierarchia pripojená jeden krát pre každú postavu. Pre každú postavu by mal byť preto vytvorený nový kontext, ktorý umožní serveru postáv udržiavať stav príslušnej postavy. Pri pripojení hierarchie do scény sa kontext uvedie ako jeden z parametrov.

2.1.3 Grafické používateľské rozhranie

Nebula Device poskytuje systém pre vstavané grafické používateľské rozhranie (Graphic User Interface, GUI). Tento systém podporuje vytvorenie okien (dialógov) spolu so základnými ovládacími prvkami ako sú tlačidlá, nápisy, editovacie polia, zoznamy súborov a adresárov na disku, kontextové menu a pod. Všetky triedy ovládacích prvkov majú spoločného predka (nGuiWidget). Prvky grafického rozhrania majú definovaný svoj vzhľad spolu s ďalšími zdrojmi ako sú textúry, zvuky, písmo a farby. Toto nastavenie sa nazýva jedným slovom *skin* (trieda nGuiSkin). Ovládacie prvky je možné vytvoriť buď priamo z C++ kódu alebo zo skriptu.

Na výber máme tiež kompletné funkčné dialógy: konzolu pre vstup a výstup skriptových príkazov, dialóg pre voľbu súborov, prechádzanie používaných textúr a dialóg zobrazujúci systémové informácie.

Zobrazenie GUI a jeho funkčnosť má na starosti nGuiServer. Ten pri otvorení automaticky volá skriptovú funkciu OnGuiServerOpen. Funkcia by mala byť definovaná a obsahovať prípadnú inicializáciu GUI systému. Zvyčajne sa tu vytvára a definuje skin. Tiež je vytvorené hlavné okno pomenované SystemRootWindow. Všetky okná vytvorené programátorom sú umiestnené v NOH pod hlavným oknom. Pri zavretí serveru sa volá skriptová funkcia OnGuiServerClose.

Obrázok 2.1: Ukážka GUI v Nebula Device.

2.1.4 Subsystém pre vstupné zariadenia

Vstupné zariadenia sú ovládané serverom triedy nInputServer. Na výber máme niekoľko virtuálnych vstupných zariadení:

- keyb – klávesnica,
- mouse – myš,
- relmouse – relatívna myš (vykazuje činnosť aj mimo výstupného okna),
- joy – joystick.

Tieto zariadenia pri stlačení tlačidla alebo pohybe generujú udalosti. Udalosti sú potom prevedené na textový popis stavu, tj. textový identifikátor zvolený programátorom. Napr. udalosť „stlačenie klávesu F1“ bude označená identifikátorom „showhelp“. Akonáhle dôjde k stlačeniu F1, vygeneruje sa udalosť showhelp. Aplikácia môže potom testovať výskyt týchto udalostí a reagovať na ne.

Výhodou je, že mapovanie udalostí na identifikátory je možné definovať v skripte. To umožňuje vykonávať zmeny bez zásahu do kódu aplikácie. Na druhú stranu ale nie je možné definovať zložitejšie klávesové skratky (ako napr. Alt+F4).

Nebula podporuje i viacero vstupných zariadení rovnakého typu zapojených naraz (napr. dva joysticky). Preto sa pri definovaní mapovania udáva aj číslo zariadenia. Ukážka mapovania udalostí vstupných zariadení pomocou skriptu:

```
sel ('/sys/servers/input')

call('beginmap')
call('map', "keyb0:f1.down", "showhelp")
call('map', "joy0:+x", "move_right")
call('map', "joy0:-x", "move_left")
call('map', "mouse0:button0.down", "attack")
call('map', "joy1:b1.long", "extra_attack")
call('endmap')
```

2.1.5 Prístup k systému súborov

Nebula Device obsahuje sadu objektov pre prácu so súbormi a adresárovou štruktúrou na disku (nFileServer2, nDirectory, nFile, ...). Hlavnými vlastnosťami tohto systému sú:

- Nezávislosť na hostiteľskom systéme súborov.
- Podpora služieb špecifických pre danú platformu (používajú sa funkcie Windows API namiesto štandardných z ANISI C).
- Objektovo orientovaný prístup k súborom a adresárom.
- Podpora pre virtuálne systémy súborov. To je užitočné pri práci s archívmi súborov.
- Podpora skratiek adresárov.

Skratky adresárov (*directory assigns*) umožňujú označiť absolútnu cestu k adresáru pomocou jednoduchého textového identifikátora. Prostredníctvom skratiek je potom možné ľahko zapisovať cesty k súborom a hlavne, v prípade potreby, zmeniť cieľový adresár. Použitie skratky vyzerá následovne:

```
"textury:mesto/strecha.jpg"
```

Namiesto:

```
"C:/program files/.../textury/mesto/strecha.jpg"
```

2.2 Herný engine Mutant

Herný engine Mutant nadväzuje na knižnicu Nebula Device, ku ktorej pridáva potrebné herné súčasti (herné objekty, úrovne hry, ...) spolu s rozhraním medzi týmito hernými súčasťami a knižnicou Nebula Device (jedná sa predovšetkým o špecializované servery a priestory – viď nižšie). Obsahuje tiež pridané knižnice pre umelú inteligenciu a inverznú kinematiku.

Ako skriptovací jazyk sa používa výhradne Lua (kapitola 2.5). Pre fyzikálnu simuláciu a detekciu kolízií slúži server využívajúci knižnicu OpenDE (kapitola 2.6). V plnej miere sa využíva objektová hierarchia a perzistentnosť objektov (kapitola 2.1.1). Engine neimplementuje žiadny vlastný formát súborov.

Autorom enginu je firma Napoleon Games[3]. Je určený predovšetkým pre pripravovanú 3D akčnú hru Mutant. Podrobné informácie o engine sa nachádzajú v elektronickej dokumentácií [7] vygenerovanej programom Doxygen[4] (MutantProgram\Documents\Sources\html).

2.2.1 Základné pojmy

- *Engine* – súbor tried, funkcií, skriptov a dátových zdrojov zabezpečujúci základnú funkčnosť určitej skupiny počítačových hier. V našom prípade sa jedná o knižnicu Nebula Device spolu s ďalšou vrstvou tried pracujúcich s hernými objektmi, scénami, fyzikálnou simuláciou, umelou inteligenciou a pod.

- *Herný objekt* (*game object*) – je základnou, samostatnou jednotkou scény. Herný objekt môže byť viditeľný (dom, postava, zbraň,...), alebo neviditeľný (kamera, zdroj zvuku, bariéra). Vzhľad, fyzikálne vlastnosti, rola alebo použitie objektu závisia od typu a nastavenia *prototypu*.

- *Prototyp* (*prototype*) – herné objekty sú vytvárané počas načítavania scény z prototypov. Prototyp obsahuje buď súbor *parametrov* alebo ľubovoľné množstvo podprototypov (vtedy sa jedná o tzv. *kontajner*). Neobsahuje ale parametre a podprototypy súčasne. Po vytvorení herného objektu ostáva jeho prototyp v pamäti. Prototypy sú zdieľané, v pamäti existuje vždy len jedna inštancia konkrétneho prototypu.

- *Parametre* (*parameters*) – zahŕňajú v sebe skupinu vlastností herného objektu, ktoré spolu súvisia. Existuje viacero typov parametrov. Napr. parametre nastavujúce vizuálne, fyzikálne, zvukové vlastnosti a pod. Parametre môžu byť rovnako ako prototypy organizované v *kontajneri*.

- *Kontajner* (*container*) – obsahuje jednu úroveň potomkov rovnakého typu ako je on sám. Kontajner nemôže obsahovať ďalší kontajner.

- *Tvorca* (*creator*) – je prostriedok pre vytváranie herných objektov z prototypov. Obsahuje názov prototypu spolu s vlastnosťami herného objektu, ktoré sú špecifické pre daný herný objekt (napr. pozícia). Scéna uložená na disku obsahuje

[3] http://www.napoleongames.cz
[4] http://www.stack.nl/~dimitri/doxygen

zoznam takýchto tvorcov. Po načítaní scény sa už tvorcovia ďalej nepoužívajú a sú odstránení z pamäti. Pred uložením sa zase spätne vytvoria z herných objektov a uložia do súboru.

2.2.2 Jadro enginu

Vstupným bodom aplikácie založenej na engine Mutant je trieda MUT_Engine. Tá po spustení najprv inicializuje skriptový server a spúšťa inicializačný skript (MutantGame\Bin\setup.lua). Úlohou inicializačného skriptu je vytvorenie väčšiny serverových objektov a nastavenie skratiek pre rôzne adresáre na disku.

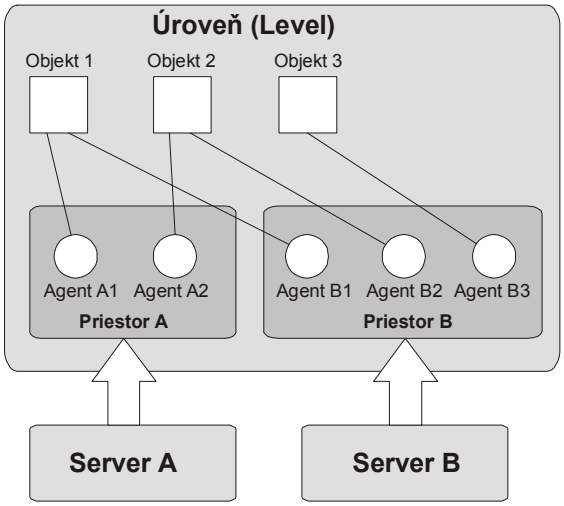
Následne je načítaný a spustený objekt triedy MUT_Game (prípadne jej potomka). Štandardne je objekt uložený v súbore MutantGame\System\game.n2. Vždy existuje iba jedna inštancia tejto triedy. MUT_Game vytvára zvyšné servery a inicializuje časti aplikácie, ktoré závisia na danom type hry. Podobne ako v MUT_Engine, i tu sa spúšťa ďalší inicializačný skript. Tento krát je to MutantGame\Scripts\System\setup_game.lua. Ten nastavuje okrem iného skin pre GUI (viď kapitola 2.1.3) a mapovanie udalostí vstupných zariadení (kapitola 2.1.4).

MUT_Game umožňuje načítať jeden alebo viac úrovní hry (levelov), zabezpečuje hlavný cyklus hry, stará sa o GUI a aktualizuje rôzne ladiace štatistiky.

2.2.3 Úrovne hry

Jednotlivé úrovne hry (levely, trieda MUT_Level) obsahujú herné objekty a sadu *priestorov* (trieda MUT_Space). Existuje napríklad vizuálny a fyzikálny priestor alebo priestor umelej inteligencie. Tieto priestory sú súčasťou levelov a spolu s nimi sa ukladajú na disk. O fungovanie priestorov sa starajú odpovedajúce servery.

Okrem priestorov sa v leveloch na disku nachádzajú aj tvorcovia (MUT_PRM_Creator). Každý tvorca počas načítavania úrovne vytvára použitím prototypu (MUT_PRM_Prototype) jeden alebo niekoľko herných objektov. Pre lepšie pochopenie pojmov viď kapitolu 2.2.1.



Obrázok 2.2: Herné objekty, agenti a priestory

MUT_Level tiež vykonáva časť hlavného herného cyklu, ktorá je závislá od daného levelu. Tu patrí aktualizácia priestorov, kamery, herných objektov a v neposlednom rade vykreslenie scény. Existuje tu i možnosť „zamrznutia“ levelu, kedy sa všetky aktualizácie dočasne zastavia.

2.2.4 Herné objekty

Typ herného objektu (trieda MUT_GO_GameObject) závisí od parametrov (MUT_PRM_Parameters) obsiahnutých v prototype. Samotný herný objekt je prvkom levelu. Všetky objekty obsahujú navyše sadu takzvaných *agentov* (MUT_Agent). Počet

a typ agentov závisí na type herného objektu. Napríklad ak má game objekt fyzikálne atribúty, potom má agenta vo fyzikálnom priestore. Agenti sú prvkami príslušných priestorov a sú teda spojnicami medzi dianím v priestore a samotným herným objektom. Agenti sú v rámci jedného objektu samostatní a nijak navzájom nespolupracujú. Komunikujú výhradne s herným objektom. Počet a druh agentov závisí od typu objektu (Obrázok 2.2).

2.3 MFC

MFC je knižnicou tried jazyka C++, poskytovanou spoločnosťou Microsoft k vytvoreniu objektovo orientovaného obalu okolo rozhrania Windows API. Verzia 6 obsahuje približne 200 tried, z ktorých niektoré sa používajú priamo a ďalšie slúžia ako základ pre vlastné triedy. Niektoré z obsiahnutých tried sú neobyčajne jednoduché. Medzi tieto triedy patrí napríklad trieda CPoint, ktorá reprezentuje bod (umiestnenie definované súradnicami x, y). Ostatné triedy sú zložitejšie. K nim možno priradiť triedu CWnd, ktorá zapuzdruje funkčnosť okna. V programe typu MFC sa zriedka volajú funkcie rozhrania API. Miesto toho sa na základe tried MFC vytvárajú objekty a volajú ich metódy.

Podrobné informácie o knižnici MFC sa nachádzajú v [9].

2.3.1 Aplikačný rámec

MFC je tiež *aplikačným rámcom*. Je nie len kolekciu tried, ale napomáha pri definovaní štruktúry celej aplikácie a stará sa o mnohé bežné funkcie v prospech danej aplikácie. Používa množstvo trikov k tomu aby sa objekty systému Windows ako okná, dialógové okná či ovládacie prvky chovali ako objekty C++. V aplikácií vytvorenej pomocou MFC formuláre a dialógové okná odosielajú nespracované hlásenia späť ovládacím prvkom, ktoré ich pôvodne vyslali. To umožňuje vytvoriť celkom samostatné a prenositeľné triedy ovládacích prvkov.

Základným kameňom aplikačného rámca MFC je architektúra dokument/pohľad, ktorá definuje štruktúru programu. Tá sa opiera o objekt Dokument, ktorý uchováva dáta aplikácie, a o objekt Pohľad, ktorý sprostredkováva pohľady na tieto dáta. Celý rad ďalších tried pracuje v spojení s dokumentom a pohľadom. Tento postup nie je zďaleka tak obmedzujúci, ako by sa na prvý pohľad mohlo zdať. Dokumentom sa môže stať aj pole bajtov, ktoré uchováva pozície na šachovnici v šachovom programe.

2.3.2 Objekt aplikácie

Jadrom aplikácie pri použití MFC je *objekt aplikácie*, odvodený od triedy CWinApp. Táto trieda poskytuje cyklus správ, ktorá správy vyhľadáva a odosiela ich do okna aplikácie. Obsahuje tiež virtuálne funkcie, ktoré je možné potlačiť a prispôsobiť tak chovanie programu vlastným potrebám. Aplikácia môže obsahovať iba jeden takýto objekt, ktorý musí byť navyše deklarovaný ako objekt s globálnym rozsahom oboru platnosti tak, aby mohol byť vytvorený v pamäťovom priestore na samom počiatku programu.

Objekt aplikácie je jediná potrebná vec pre spustenie programu. Nepoužíva sa tu žiadna funkcia main ani WinMain. Chod programu automaticky zabezpečuje funkcia AfxWinMain definovaná už v samotnom MFC. Tá používa objekt aplikácie, ktorý práve

preto musí byť deklarovaný ako globálny. Ihneď po spustení aplikácie sú volané metódy triedy CWinApp: InitApplication, InitInstance a Run. Pri ukončení funkcia ExitInstance.

Virtuálna funkcia InitInstance neobsahuje žiadny funkčný kód. Je však ideálna pre nastavenie všetkých počiatočných hodnôt. Prinajmenšom pre vytvorenie okna, ktoré bude reprezentovať aplikáciu na obrazovke. V prípade že sa použije pre alokovanie pamäťového priestoru, je funkcia ExitInstance vhodným miestom, kde sa pridelené prostriedky uvoľnia.

Funkcia Run obsahuje hlavý cyklus pre spracovanie správ. Táto funkcia sa môže potlačiť pri potrebe prispôsobiť cyklus správ a nahradiť ho vlastným.

2.3.3 Objekty okien

Trieda CWnd a jej potomkovia poskytujú objektovo orientované rozhranie okna alebo okien vytváraných aplikáciou. Trieda CFrameWnd, odvodená od CWnd, definuje chovanie *rámcových okien*. Rámcové okno je najvyššie postavené okno, ktoré zaisťuje základné komunikačné rozhranie aplikácie s vonkajším svetom. V širšom kontexte architektúry dokument/pohľad má rámcové okno väčší význam. Predstavuej inteligentný kontajner, do ktorého sú umiestňované pohľady, panely nástrojov, stavové riadky a mnoho ďalších objektov používateľského rozhrania. Aplikácia vytvára okno tak, že vytvorí príslušný objekt a zavolá jeho funkciu Create alebo CreateEx. Tie poskytujú široký výber možností pre nastavenie vlastností okna.

2.3.4 Mapa správ

Mapa správ je ja tabuľka, ktorá zlaďuje správy systému Windows a členské funkcie objektov. Vo chvíli, keď hlavné okno aplikácie obdrží správu, MFC prejde mapu správ, zistí, ktorá obslužná rutina je určená pre spracovanie správy, a tu potom zavolá. Mapa správ je metódou, ktorou sa MFC vyhýba rozvláčnemu rozhraniu tabuliek virtuálnych funkcií triedy (vtable), ktorá by sa musela použiť, keby každá trieda obsahovala vlastnú virtuálnu funkciu pre každú potencionálnu správu (je treba si uvedomiť, že systém Windows používa niekoľko stoviek typov správ).

Všetky triedy odvodené od triedy CCmdTarget môžu obsahovať mapu správ. To, ako MFC vnútorne implementuje mapy správ, zostáva ukryté za pomerne zložitými makrami, ale používanie je jednoduché (viď ukážka nižšie):

- Uvedenie príkazu DECLARE_MESSAGE_MAP v deklarácii triedy definuje mapu správ.
- Mapa správ sa implementuje pridaním makier rozpoznávajúcich správy, ktoré bude trieda ošetrovať, medzi príkazy BEGIN_MESSAGE_MAP a END_MESSAGE_MAP.
- Nakoniec sa pridajú členské funkcie pre ošetrenie správ uvedené identifikátorom afx_msg. Tento identifikátor je iba viditeľnou pripomienkou toho, že sa jedná o obslužné rutiny správ. Môže sa i vynechať.

Mapy správ sú dedené rovnako ako všetky ostatné členy triedy.

Nasleduje ukážka aplikácie typu MFC, ktorá vytvorí rámcové okno s nápisom „Ahoj svet.“, zarovnaným do stredu okna.

```
#include <afxwin.h>             // vlozenie kniznice mfc
```

```
class CAplikacia : public CWinApp {
public:
    virtual BOOL InitInstance();
};

class CHlavneOkno : public CFrameWnd {
protected:
    afx_msg void OnPaint();
    DECLARE_MESSAGE_MAP()
}

CAplikacia aplikacia;           // deklaradia objektu aplikacie

BOOL CAplikacia::InitInstance() {
    m_pMainWnd = new CHlavneOkno;
    m_pMainWnd->Create(NULL, "Hlavne okno");
    m_pMainWnd->ShowWindow(m_nCmdShow);
    m_pMainWnd->UpdateWindow();
    return TRUE;
}

BEGIN_MESSAGE_MAP (CHlavneOkno, CFrameWnd)
    ON_WM_PAINT()
END_MESSAGE_MAP()

CHlavneOkno::OnPaint() {
    CPaintDC dc(this);
    CRect rect;

    GetClientRect(&rect);
    dc.DrawText("Ahoj svet"), -1, &rect,
        DT_SINGLELINE | DC_CENTER | DT_VCENTER);
}
```

2.3.5 Súbor zdrojov

Súbor zdrojov je textový súbor typu skript, ktorý definuje zdroje aplikácie. Súbor má podľa zavedených konvencií príponu .rc (z toho sa tiež nazýva súborom RC). *Zdroj* je binárnym objektom. Systém Windows podporuje niekoľko typov zdrojov vrátane ponúk, ikon, rastrových obrázkov, reťazcov či kompletných dialógov s ovládacími prvkami. Prekladač súborov zdrojov (Rc.exe), ktorý je dodávaný spolu s balíkom Windows Software Development Kit (SDK), prekladá príkazy v súbore RC do strojového jazyka a pripája výsledné zdroje ku spustiteľnému súboru aplikácie. Každý zo zdrojov je určovaný na základe reťazca alebo celočíselného identifikátoru. Z kódu aplikácie je potom možné vytvárať príslušné objekty jednoduchým načítaním zo súboru zdrojov.

2.3.6 Ďalšie objekty

Knižnica MFC ponúka triedy pre ďalšie objekty aplikácie ako sú ponuky (trieda CMenu), dialógy (CDialog), panely nástrojov (CToolBar) a podobne. Tieto objekty je možné dopredu pripraviť v súbore zdrojov alebo vytvárať manuálne počas behu aplikácie.

2.4 XML

Jazyk XML (Extensible Markup Language) bol štandardizovaný konzorciom W3C[5] a jeho základom sa stala najpoužívanejšia podmnožina jazyka SGML (Standard Generalized Markup Language), ktorá bola ďalej obmedzená pevne určenými parametrami a prísnou syntaxou. Popisuje triedu dátových objektov nazývaných XML dokumenty a čiastočne popisuje správanie počítačových programov, ktoré ich spracovávajú.

Je určený predovšetkým pre ukladanie, predávanie a publikovanie semi-štruktúrovaných dát. Jedná sa v podstate o metajazyk, čo znamená, že je určený pre popis ďalších jazykov. XML poskytuje mechanizmus pre uloženie obmedzení obsahu, vzájomných vzťahov a logickej štruktúry.

Podrobné informácie o jazyku XML sa nachádzajú v [3], [4] alebo [5].

2.4.1 Elementy

Základnými prvkami jazyka sú *elementy*, ktoré sú do seba vzájomne vnorované. Elementy sú určené pomocou *značiek* (*tags*), pričom väčšine elementov odpovedajú dve značky – počiatočná a koncová. Výnimkou je prázdny element, ktorý je určený iba jednou značkou. Značky sú uzavreté medzi znaky ‘<’ a ‘>’, koncová značka má navyše na začiatku znak ‘/’, značka pre prázdny element má znak ‘/’ na konci.

```
<kniha>
      <názov>Päť týždňov v balóne</názov>
      <autor>Jules Verne</autor>
      <obrázok />
</kniha>
```

Každý neprázdny element musí byť uzavretý medzi obidve značky a toto uzavretie musí byť správne uzátvorkované. Ďalej musí byť celý XML dokument uzavretý v jenom koreňovom elemente.

2.4.2 Atribúty

Ďalšími prvkami XML dokumentu sú *atribúty*. Nachádzajú sa vo vnútri počiatočnej značky elementu a obsahujú dodatočné informácie vzťahujúce sa k danému elementu. Atribúty majú nasledujúcu formu: názov=“hodnota“.

```
<kniha typ="dobrodružná" cena="120">
      <názov>Päť týždňov v balóne</názov>
      <autor>Jules Verne</autor>
      <obrázok src="jules_verne/pat_tyzdnov_v_balone.jpg"/>
</kniha>
```

2.4.3 XML deklarácia

Na začiatku každého XML dokumentu by mala byť umiestnená tzv. *XML deklarácia*. Jedná sa o element obsahujúci informácie o používanej verzii XML, použitej znakovej

[5] W3C (World Wide Web Consortium) – http://www.w3.org

sade a informáciu o tom, či je pre správnu interpretáciu obsahu nutné použiť externe definované deklarácie značiek.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

2.4.4 Ďalšie prvky XML dokumentu

V XML dokumente sa môžu ďalej nachádzať komentáre, sekcia CDATA a inštrukcie pre spracovanie.

Komentár je ľubovoľný text uzavretý medzi znakmi <!-- a -->. Nepodlieha spracovaniu dokumentu. Sekcia CDATA je vhodná pre prípad, kedy potrebujeme do dokumentu vložiť väčší kus textu obsahujúceho znaky so špeciálnym významom (‘<’, ‘>’, ‘&’ a pod.). Vo vnútri tejto sekcie, tj. medzi znakmi <![CDATA[a]]>, je tento význam ignorovaný. Inštrukcie pre spracovanie sú príkazy v tvare <?*identifikátor dáta*?>. Sú učené pre nadradený program a nepodliehajú spracovaniu dokumentu.

2.5 Lua

Lua je rozširujúci programovací jazyk navrhnutý pre podporu procedurálneho programovania s uľahčeným popisom dát. Ponúka i podporu pre objektovo orientované a fukcionálne programovanie. Jej zámerom je výkonný a jednoduchý konfiguračný jazyk pre programy písané v jazyku C. Lua teda nemá koncepciu hlavného programu ale funguje vložená do hostiteľského programu. Hostiteľský program môže volať funkcie pre spustenie Lua kódu, zapisovať a čítať Lua premenné a registrovať C funkcie pre volanie z Lua kódu.

Distribučný balíček je voľne šíriteľný, obsahuje Lua knižnicu i samostatný interpreter skriptov. Podrobné informácie o jazyku Lua sa nachádzajú v manuáli [2].

2.5.1 Lexikálne konvencie

Identifikátory v jazyku Lua môžu obsahovať alfanumerické znaky a podčiarniky, nesmú začínať číslicou. Ako identifikátory nesmú byť použité kľúčové slová:

```
and       break    do        else      elseif    end
false     for      function  if        in        local
nil       not      or        repeat    return    then
true      until    while
```

Lua rozlišuje veľkosť písmen. Podľa konvencie, idenfikátory začínajúce podčiarnikom nasledovaným veľkými písmenami sú rezervované pre interné premenné. Nasledujúre reťazce označujú ostatné tokeny:

```
+     -     *     /     ^     =
~=    <=    >=    <     >     ==
(     )     {     }     [     ]
;     :     ,     .     ..    ...
```

Textové reťazce môžu byť uzavreté v dvojici apostrofov alebo úvodzoviek. Môžu obsahovať špeciálne znaky uvedené znakom ‘\’ podobne ako v jazyku C. Napr. \n, \t, \”, \’ alebo \\. Navyše môžu byť reťazce uzavreté v dvojitých hranatých zátvorkách [[…]].

Reťazce v tomto prípade môžu presahovať cez niekoľko riadkov a obsahovať vnorené hranaté zátvorky. Neinterpretujú sa žiadne špeciálne znaky.

Jednoriadkové komentáre začínajú dvojicou pomlčiek --. Viacriadkové komentáre podobne dvojicou pomlčiek nasledovanou dvojitými hranatými zátvorkami --[[...]]. Na prvom riadku môže byť prvým znakom aj #. To umožňuje použitie jazyka Lua ako interpretra v Unixových systémoch.

```
-- jednoriadkovy komentar
--[[
    viacriadkovy
    komentar
]]
```

2.5.2 Hodnoty a typy

Lua je jazyk s dynamickými typmi. To znamená že typ nie je vlastnosťou premenných ale vlastnosťou hodnôt. Neexistujú žiadne definície typov. Každá hodnota obsahuje svoj vlastný typ.

Jazyk obsahuje osem základných typov: *nil* – typ hodnoty nil, *boolean* – typ hodnôt false a true, *number* – reprezentuje reálne číslo, *string* – pole ľubovoľných 8-bitových znakov, *function* – referencia na fukciu, *userdata* – referencia na používateľské dáta definované a menené výlučne v jazyku C, *thread* – reprezentuje nezávislé vlákno s bežiacim Lua kódom, *table* – referencia na tabuľku.

Tabuľky sú implementované ako asociatívne heterogénne polia. To znamená, že indexom tabuľky môže byť ľubovoľná hodnota okrem hodnoty nil a navyše môžu tabuľky obsahovať ľubovoľné hodnoty okrem nil.

Medzi reťazcami a číslami prebieha automatická konverzia. Formát čísla pri prevode na reťazec je možné definovať. Hodnoty nil a false sú považované za nepravdu, ostatné hodnoty (vrátane čísla nula a prázdneho reťazca) za pravdu.

2.5.3 Premenné

Rozlišujeme tri typy premených: globálne, lokálne a polia tabuliek. Premenné (ako aj parametre fukcie) sa deklarujú ich menom. Pred prvým priradením majú hodnotu nil. Štandardne sú globálne, pokiaľ nie sú explicitne deklarované pomocou kľúčového slova local. Pre zadávanie indexov tabuliek sa používajú hranaté zátvorky alebo bodka:

```
pole["index"] = hodnota
pole.index = hodnota
```

2.5.4 Príkazy

Lua podporuje takmer všetky bežné príkazy ako Pascal alebo C: priradenia, kontrolné štruktúry, volania funkcií, konštruktory tabuliek a deklarácie premenných. Príkazy môžu byť ukončené bodkočiarkou.

Spustiteľná jednotka Lua kódu sa nazýva *dávka* (*chunk*). Dávka je postupnosť príkazov, ktoré sú sekvenčne vykonávané. Lua narába s dávkou ako s telom anonymnej

funkcie. Dávka môže teda obsahovať lokálne premenné a návratové hodnoty. Je uložená buď v súbore alebo reťazci hostiteľského programu. Spustená dávka je najprv predkompilovaná do medzikódu, ktorý je následne interpretovaný. Takýto medzikód je možné pre optimalizáciu vygenerovať do binárneho súboru a priamo spúšťať.

Príkazy môžu byť zoskupené v bloku. To umožňuje napríklad kontrolovať platnosť lokálnych premenných.

```
do
    prikaz
    ...
end
```

Lua povoľuje viacnásobné priradenia. Premenné a hodnoty sú oddelené čiarkami.

```
premenna1, premenna2, pole[index] = hodnota1, hodnota2, funcia()
```

Kontrolné štruktúry if, while a repeat majú bežný význam a intuitívnu syntax.

```
while vyraz do
    prikaz
    ...
end

repeat
    prikaz
    ...
until vyraz

if vyraz then
    prikaz
    ...
elseif vyraz     -- volitelna cast
    prikaz
    ...
else             -- volitelna cast
    ...
end
```

Príkaz for má dve formy: numerickú a univerzálnu. Numerická forma opakuje blok príkazov pokým kontrolná premenná neprekročí maximálnu hodnotu.

Univerzálna forma používa pre výpočet hodnoty kontrolnej premennej zadanú iteračnú funkciu a stavovú hodnotu. Počiatočnú hodnota sa zadáva iba pre prvú kontrolnú premennú. Iteračná funcia má dva parametre: stav a prvú kontrolnú premennú. Vracia hodnoty pre všetky kontrolné premenné. Cyklus končí ak iteračná funkcia vráti nil.

```
for premenna = pociatocna_hodnota, maximalna_hodnota, krok do
    prikaz
    ...
end

for premenna1, premenna2, ... in funkcia, stav, poctiatoc_hodn do
    prikaz
```

```
    ...
end
```

Pre ukončenie najvnútornejšieho cyklu sa používa príkaz break. Príkaz return slúži pre ukončenie a návrat z funkcie. Tieto príkazy musia stáť na konci bloku pred kľúčovým slovom end.

Lokálne premenné môžu byť deklarované kdekoľvek vo vnútri bloku. Deklarácia môže obsahovať inicializačné priradenie. V opačnom prípade majú premenné hodnotu nil.

```
local premenna1, premenna2, ... = vyraz1, vyraz2, ...
```

2.5.5 Výrazy

Základnými výrazmi sú čísla, textové reťazce, premenné, definície funkcií, volania funkcií a tabuľkové konštruktory. Základné výrazy môžu byť prepojené operátormi. Výsledkom výrazu uzatvoreného v zátvorkách je vždy jedna hodnota. Napr. výsledkom výrazu (f(x, y, z)) je prvá návratová hodnota funkcie alebo nil.

Operátory delíme na:

- aritmetické: + (sčítane), - (odčítanie), * (násobenie), / (delenie), ^ (umocňovanie), unárne – (negácia)
- relačné: == (rovné), ~= (nerovné), < (menšie), > (väčšie), <= (menšie alebo rovné), >= (väčšie alebo rovné)
- logické: and (konjunkcia), or (disjunkcia), not (negácia). Logické operátory používajú skrátené vyhodnocovanie.
- reťazcové: .. (konkatencia)

Tabuľkové konštruktory sú výrazy, ktoré vytvárajú tabuľky. Sú uzatvorené v zložených zátvorkách {...}, v ktorých sa nachádza zoznam inicializačných hodnôt.

```
tabulka1 = {hodnota1, hodnota2, hodnota3}
tabulka2 = {[0] = hodnota1, [1] = hodnota2, [2] = hodnota3}
tabulka3 = {hodnota1, ["index2"] = hodnota2, index3 = hodnota3}
```

Volanie funkcií má tvar: *prefix*(*zoznam_argumentov*), kde *prefix* je výraz s hodnotou typu funkcia. Volanie *objekt*:*funcia*(...) je skratkou pre *objekt*.*funkcia*(*objekt*, ...) a používa sa pre volanie „metód objektu“ (v skutočnosti sú to funkcie v tabuľke).

```
funkcia1()
premenna1, premenna2 = objekt:funkcia2(parameter1, funkcia3())
```

Definície funkcií sú označené kľúčovými slovami function, end.

```
function menofunkcie1(parameter1, parameter2, ...)
    prikaz1
    ...
end
```

Nasledujúce dva zápisy sú ekvivalentné.

```
local function menoobjektu.menofunkcie()
   ...
end

local menoobjektu
menoobjektu["menofunkcie"] = function()
   ...
end
```

2.5.6 Manažment pamäte

Lua prevádzkuje automatický manažment pamäte. Programátor sa nemusí starať o žiadne alokovanie a uvoľňovanie pamäte. Vstavaný garbage collector sa spúšťa sám podľa potreby, prípadne je možné vyvolať jeho spustenie z jazyka C.

2.6 Open Dynamics Engine

Open Dynamics Engine (ODE) je voľne šíriteľná knižnica pre fyzikálnu simuláciu tuhých telies prepojených rôznymi typmi kĺbov. Je vhodná napríklad pre simuláciu dynamiky vozidiel, robotov a iných pohybujúcich sa objektov v prostredí virtuálnej reality. Obsahuje vstavanú detekciu kolízií. Autorom ODE je Russell Smith spolu s viacerými prispievateľmi.

Podrobné informácie o fungovaní a používaní ODE obsahuje manuál [1].

2.6.1 Základné vlastnosti ODE

ODE je vhodná pre simuláciu kĺbových štruktúr. Takéto štruktúry sú zložené z *tuhých telies* (*rigid bodies*) rôznych povrchov, ktoré sú navzájom prepojené kĺbmi rôznych typov. Je navrhnutá pre použite v interaktívnej simulácii prebiehajúcej v reálnom čase (*real-time*). Obzvlášť je vhodná pre simuláciu pohybujúcich sa objektov v meniacom sa prostredí virtuálnej reality. Používateľ má kompletnú voľnosť pri menení štruktúr systému i počas vlastnej simulácie. ODE kladie veľký dôraz na rýchlosť a stabilitu spolu s fyzikálnou presnosťou.

ODE obsahuje vstavaný systém pre detekciu kolízií. Tento systém nemusí byť nutne použitý. Programátor má možnosť doplniť vlastnú detekciu kolízií. Kolízny systém poskytuje rýchlu identifikáciu potencionálnych kolízií pomocou konceptu tzv. priestorov.

Pri kolízii vzniká medzi telesami špeciálny nepenetračný kolízny kĺb. ODE teda používa tzv. pevné kontakty. Počas simulácie sa ale bežne stáva, že teleso pri kontakte čiastočne prenikne do iného telesa. To je dôsledok konštantnej dĺžky kroku simulácie. V takomto prípade mechanizmus pre znižovanie chýb simulácie vytlačí preniknuté teleso von.

Súčasne môže existovať niekoľko fyzikálnych *svetov* (*world*). Simulácia prebieha v každom svete samostatne. Objekty v jednom svete nijakým spôsob neinteragujú s objektmi v inom svete.

Zoznam hlavných vlastností ODE:

- Tuhé telesá s ľubovoľnou distribúciou hmoty.
- Typy kĺbov: guľa a jamka, pánt, pánt 2, hranolový jazdec, fixný, kontaktný, uhlový motor, univerzálny.
- Kolízne primitíva: sféra, kváder, kapsula, rovina, lúč, trojuholníková sieť.
- Kolízne priestory: jednoduchý, hašovaný, s použitím štruktúry Quad-tree.
- Simulačná metóda: rovnice pohybu sú odvodené z modelu založného na Lagrangeovom multiplikátore rýchlosti.
- Výber dĺžky kroku: obidve štandardné metódy veľkých matíc (big matrix), nová iteračná metóda QuickStep.
- Kontakty a model trenia: založené na metóde Dantzig LCP opísanej Baraffom, ODE tiež implementuje rýchlejšiu aproximáciu podľa Columbovho modelu trenia.
- Obsahuje rozhranie pre jazyk C i C++.
- Optimalizácie špecifické pre rôzne platformy.

2.6.2 Tuhé telesá

Tuhé telesá majú z hľadiska fyzikálnej simulácie rôzne vlastnosti. Niektoré z nich sa menia s časom. Sú to:

- Vektor pozície referenčného bodu telesa. V aktuálnej verzii ODE je referenčný bod totožný s ťažiskom.
- Lineárna rýchlosť (linear velocity) referenčného bodu.
- Orientácia telesa reprezentovaná kvaterniónom alebo maticou otočenia.
- Uhlová rýchlosť (angular velocity) určujúca ako rýchlo sa mení orientácia v čase.

Ostatné vlastnosti sa obyčajne s časom nemenia. Jedná sa o vlastnosti definujúce hmotu telesa:

- Hmotnosť telesa.
- Pozícia ťažiska hmoty.
- Matica zotrvačnosti. Je to matica veľkosti 3x3, ktorá určuje ako je hmota telesa distribuovaná okolo ťažiska.

Je treba si uvedomiť, že povrch tuhého telesa je definovaný zvlášť a teda nie je vlastnosťou dynamiky telesa. Využíva sa iba pre potreby detekcie kolízií.

Skupiny telies, ktoré sú navzájom prepojené kĺbmi vytvárajú tzv. ostrovy. S ostrovmi sa počas simulácie pracuje samostatne.

Každé teleso môže byť aktívne alebo neaktívne. Neaktívne telesá sú vynechané z fyzikálnej simulácie, čo môže zvýšiť výkonnosť systému. Ostrov je aktívny v prípade ak

je aspoň jedno teleso v tomto ostrove aktívne. Ak dôjde ku kolízii s neaktívnym telesom, toto teleso sa stane automaticky aktívnym.

K telesu je možné pripojiť ľubovoľné používateľské dáta.

2.6.3 Kĺby

Kĺb (*joint*) znamená vzťah medzi dvoma telesami, ktorý určuje obmedzenia pre vzájomnú polohu a orientáciu týchto telies. Jednotlivé kĺby sú pri bežnom nastavení pevné. To znamená, že obmedzenia definované kĺbmi sú striktné a nesmú byť porušené. V prípade že nastane chyba, mechanizmus redukovania chýb sa okamžite snaží opraviť polohu telies.

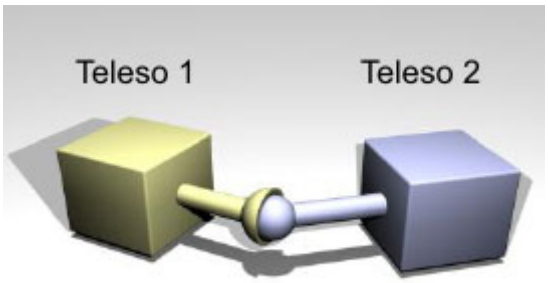
V praxi sa ale stretávame i s mäkkými kĺbmi, pri ktorých je možné presiahnuť hranice. Mäkké kĺby sú praktické napr. pre simuláciu pružinového spojenia medzi telesami, alebo simuláciu kolízií mäkkých materiálov. ODE umožňuje nastaviť mäkkosť každého kĺbu.

Kĺby môžeme zaraďovať do skupín a potom hromadne odoberať z fyzikálneho sveta. To je veľmi užitočné pri práci s kontaktnými kĺbmi, ktoré sa pridávajú a odoberajú v každom kroku simulácie.

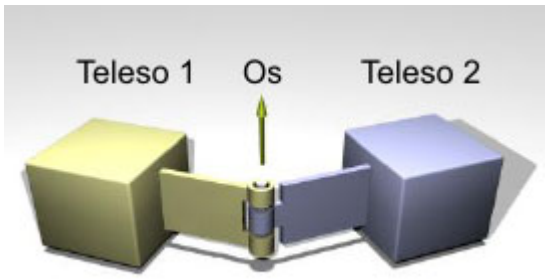
Ku kĺbu je možné pripojiť ľubovoľné používateľské dáta.

Existuje viacero typov kĺbov:

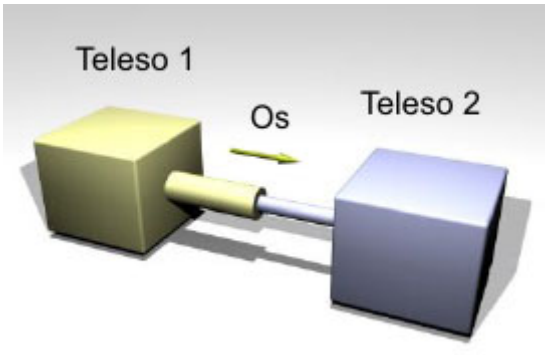
- Guľa a jamka. Viď Obrázok 2.3.
- Pánt. Obrázok 2.4.
- Jazdec. Obrázok 2.5.
- Univerzálny kĺb. Obrázok 2.6.
- Pánt 2. Obrázok 2.7.
- Kontaktný kĺb. Obrázok 2.8. Zabraňuje prieniku telies v bode kontaktu. A to tým spôsobom, že ku telesám pridá odpudivé sily v smere normály kontaktu. Tieto kĺby sa štandardne vytvárajú a rušia v každom kroku fyzikálnej simulácie ako reakcia na detekciu kolízií. Navyše je možné, pridaním špeciálnych síl kolmých na normálu, simulovať trenie v mieste kontaktu.
- Fixný kĺb. Zabezpečuje fixnú polohu medzi dvoma telesami.
- Uhlový motor. Tento kĺb sa používa pre kontrolu relatívnych uhlových rýchlostí medzi dvoma telesami. Umožňuje zadať krútiaci moment v každej osi otáčania, prípadne obmedzenia pre vzájomnú orientáciu telies. Typicky sa používa v kombinácii s kĺbom iného typu (napr. s kĺbom typu guľa a jamka).



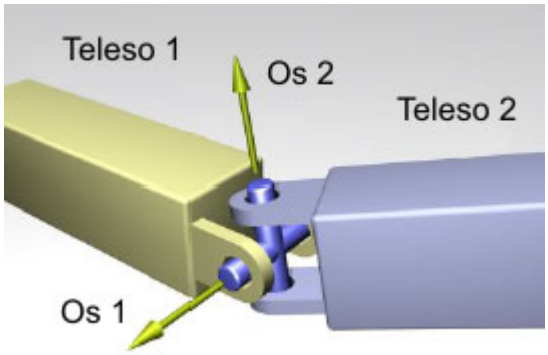
Obrázok 2.3: KÍb typu guľa a jamka.



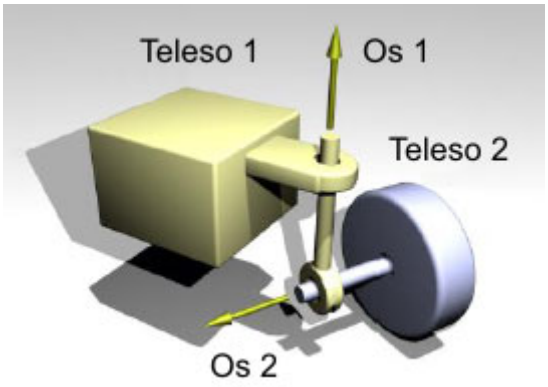
Obrázok 2.4: KÍb typu pánt.



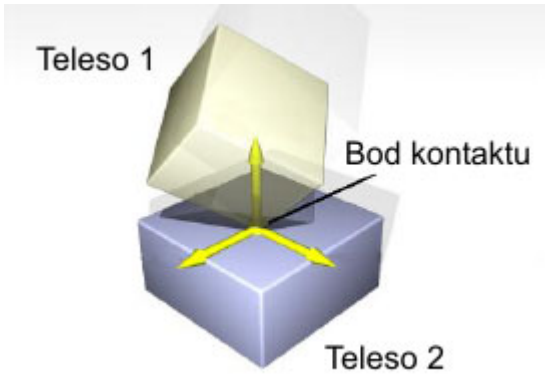
Obrázok 2.5: KÍb typu jazdec.



Obrázok 2.6: Univerzálny kÍb.



Obrázok 2.7: KÍb typu pánt 2.



Obrázok 2.8: Kontaktný kÍb.

2.6.4 Detekcia kolízií

Samostatnou časťou ODE je systém pre detekciu kolízií. Využitie tohto systému je voliteľné. Rovnako dobe môže byť použitý aj alternatívny kolízny systém.

Pre detekciu kolízií je potrebné definovať povrch jednotlivých objektov. Na začiatku každého kroku simulácie sú jednotlivé povrchy testované a výsledkom je zoznam kontaktov medzi objektmi. Kontakt v sebe nesie informáciu o pozícii kontaktného bodu, normále, hĺbke penetrácie a dvojicu identifikátorov povrchov. Bežne medzi dvoma povrchmi existuje súčasne viacero kontaktov. Zoznam kontaktov môže používateľ ďalej spracovať a nakoniec sám vytvorí a pridá do fyzikálneho sveta kontaktné kĺby. Na konci kroku (po samotnej simulácii) je potrebné kontaktné kĺby odstrániť.

Povrchy používané v ODE sú tuhé a môžu mať rôzne tvary. Na výber máme tieto: sféra, kváder, kapsula, rovina, lúč, trojuholníková sieť. Programátor má tiež možnosť definovať povrch s vlastnou geometriou. Povrchové objekty obsahujú odkaz na asociované tuhé teleso. Vďaka tomu môže kolízny systém aktualizovať pozície a otočenie povrchových objektov v priestore. Okrem toho môžu povrchové objekty existovať aj samostatne, bez prepojenia na tuhé teleso. To je dobre využiteľné v prípade statických objektov ako sú napr. budovy, cesty a podobne.

Skupina povrchov môže byť spoločne umiestnená v jednom *priestore* (*space*). Priestor je určený pre urýchlenie detekcie kolízií. Prebieha v ňom špeciálny typ orezávania pri hľadaní potencionálneho prieniku povrchov. Priestor môže obsahovať i ďalšie priestory. To je užitočné pre rozdelenie celého prostredia do hierarchickej štruktúry za účelom optimalizácie rýchlosti. V takomto prípade je možné testovať i potencionálne prieniky medzi rôznymi priestormi.

Každý povrch obsahuje navyše bitové pole, pomocou ktorého je možné definovať kategórie povrchov a bitové pole povolených kategórií. Tieto polia asistujú priestoru pri vyhodnocovaní, ktoré povrchy môžu navzájom interagovať a ktoré nie. Použitie tejto vlastnosti je voliteľné.

Ku povrchovému objektu je možné pripojiť ľubovoľné používateľské dáta.

Existuje niekoľko typov priestorov, ktoré sa od seba líšia používanými štruktúrami a algoritmom pre kolízne orezávanie:

- Jednoduchý priestor. Neobsahuje žiadne orezávanie, testuje všetky dvojice povrchov. Časová zložitosť hľadania kolízií medzi n objektmi je $O(n^2)$.

- Multi-rozmerný hašovaný priesotor. Používa interné dátové štruktúry, ktoré zaznamenávajú ako každý povrch zasahuje do buniek viacerých trojrozmerných mriežok. Každá mriežka má kubické bunky s dĺžkou strany 2^i, kde i je celočíselná hodnota z definovaného intervalu. Časová zložitosť je v tomto prípade O(n) (ak nie sú objekty zoskupené príliš blízko seba).

- Quad-tree priestor. Používa predpripravený hierarchický AABB strom. Je výrazne výkonný pre veľké množstvá objektov umiestnených vo svete s krajinným povrchom.

3 Rozbor riešenia

Zo zadania práce priamo vyplývajú dva prvotné problémy, respektíve dve úlohy o ktorých je potrebné najprv rozhodnúť. Prvou úlohou je zvoliť technológiu pre vytváranie grafického používateľského rozhrania (GUI). Druhou je zvoliť formát súboru pre konfiguráciu GUI a šablón.

3.1 Základné pojmy

- *Šablóna* (*template*) – popis ovládacích prvkov dialógu pre nastavovanie vlastností parametrov alebo herných objektov. Ku každému typu parametrov (herného objektu) je priradená práve jedna šablóna. Pri označení parametrov/objektu editor zobrazí v zvolenom dialógu ovládacie prvky definované v odpovedajúcej šablóne a naplní ich aktuálnymi hodnotami.

- *Konfigurácia editora* – súbor konfiguračných súborov, šablón a funkčných skriptov definujúcich výzor GUI a fungovanie editora pre daný typ hry.

3.2 Voľba technológie pre GUI

Zamerajme sa najprv na prvú úlohu. Pripomeňme, že podľa zadania aplikácia vyžaduje hlavnú ponuku (v podobe roletových ponúk alebo ikon) a tri základné dialógy:

- zoznam herných objektov v scéne s filtrovaním a vyhľadávaním,
- stromový prehľad otvoreného prototypu s tlačidlami pre manipuláciu s parametrami,
- a dialóg s ovládacími prvkami pre nastavovanie vlastností herného objektu alebo parametrov podľa zvolenej šablóny.

Na ich vytvorenie budeme potrebovať okrem samotných okien (dialógov) tieto ovládacie prvky: statické textové pole, tlačidlá s obrázkom/textom, editačné pole, zoznam s položkami (kde položky sú odlíšiteľné farebne alebo ikonami), stromový zoznam s položkami (kde položky sú opäť odlíšiteľné farebne alebo ikonami), tabuľku a prípadne aj roletové ponuky, pole s vysúvacím zoznamom položiek a políčko pre zaškrtávanie.

Do úvahy prichádzajú dve technológie: MFC (kapitola 2.3) a systém pre GUI vstavaný v Nebule (kapitola 2.1.3). Tieto technológie sú diametrálne veľmi odlišné. Prvky v Nebula GUI sú priamo potomkami triedy nRoot a sú teda súčasťou objektovej hierarchie. Môžu byť vytvárané a ovládané ako z C++, tak zo skriptu. Tu by bolo možné dobre využiť i perzistentnosť objektov. Nebula GUI je vykresľované vo vnútri výstupného okna. Nevýhodou je nutnosť ručne definovať skin pre všetky okná a ovládacie prvky. Ide nie len o veľkosti a pozície, ale aj o typ písma, farby, textúry a mapovanie textúr. Navyše pri krátkom testovaní zistíme, že dialógy vykazujú chybné správanie pri zmene veľkosti ako ukazuje obrázok na ďalšej strane. Dialógy tiež nepodporujú automatické spracovanie klávesy Tab. Zdá sa, že toto GUI je primárne určené pre jednoduché statické obrázkové prvky, ovládané prevažne pomocou myši, aké sú bežne potrebné v počítačových hrách. Zložitejšie prvky a dialógy slúžia najme pre ladenie a nie je im venovaná taká pozornosť.

Obrázok 3.1: Chybné správanie dialógu v Nebula Device

Naproti tomu, pomocou MFC, je možné vytvoriť dialógy mimo hlavného okna, so štandardným výzorom a fungovaním ako sme zvyknutí zo systému Windows. Základné dialógy môžu byť navrhnuté priamo vo vizuálnom editore. Nevýhodou je potreba doplniť nadstavbu pre všetky dialógy a ovládacie prvky so skriptovým rozhraním.

Poslednou rozhodujúcou otázkou je dostupnosť všetkých potrebných ovládacích prvkov. Knižnica MFC samozrejme prichádza s paletou všetkého, čo kedy bude potrebné. Použitie Nebula GUI by si vyžadovalo doprogramovanie obyčajného aj stromového zoznamu s farebne odlíšiteľnými položkami a zrejme i úpravu textovej tabuľky (chýba označovanie raidkov/buniek). Navyše môže v budúcnosti vzniknúť potreba ďalších ovládacích prvkov, ktoré v Nebula GUI nie sú (napr. pole s vysúvacím zoznamom položiek).

Vzhľadom na obmedzené použite a potencionálnu chybovosť Nebula GUI bolo zvolené rozhranie MFC. Riešenie bude vyzerať nasledovne:

- Aplikácia vytvorí prostredníctvom MFC systémové okno, v jeho klientskej oblasti sa bude nachádzať okno s grafickým výstupom enginu Nebula Device.
- Pre jednotlivé typy hotových dialógov, ponúk a ovládacích prvkov budú existovať triedy so skriptovým rozhraním, ktoré pomocou MFC vytvoria príslušný objekt GUI.
- Udalosti vniknuté používaním ovládacích prvkov (napr. stlačenie tlačidla alebo voľba z ponuky) budú môcť byť prepojené so skriptovými funkciami.
- Šablóny budú využívať jeden spoločný dialóg pre zobrazenie svojich ovládacích prvkov.

Ako sa neskôr ukázalo, ani toto riešenie nebolo úplne priamočiare.

3.3 Voľba formátu konfiguračných súborov GUI

Druhou úlohou je voľba formátu súborov pre konfiguráciu grafického používateľského rozhrania a šablón. Súbory by mali byť textové a ľahko upraviteľné bez použitia ďalšieho nástroja. Nie je potrebné ukladanie týchto súborov z editora. Do úvahy prichádza technológia XML (viď kapitolu 2.4) alebo skriptovací jazyk Lua (kapitola 2.5).

XML dokumenty prinášajú so sebou štandardizovanú podobu pre ukladanie dátových zdrojov. Dáta je možné prehľadne hierarchicky usporiadať, čo uľahčuje ich manuálnu

úpravu. Textové názvy značiek a ich parametrov zvyšujú čitateľnosť dokumentu. XML navyše umožňuje prídavnými technikami definovať štruktúru a povolený obsah dokumentu. Existuje viacero dostupných knižníc pre načítavanie a validovanie XML dokumentov. Ani enginu Nebula Device nie je táto technológia neznáma. Obsahuje priamo v sebe zabudovanú knižnicu TinyXML[6], pre jednoduchú a rýchlu prácu so XML dokumentmi.

Nevýhodou použitia XML je potreba vytvoriť špecializovaný mechanizmus pre načítavnie rôznych typov konfiguračných súborov (šablóny, dialógy, ponuky a pod.).

Konfigurácia GUI a šablón zadaná pomocou skriptu je menej štandardná, no o to zaujímavejšia alternatíva. V podstate sú možné dva prístupy. Buď využiť perzistenciu objektov typickú pre engine Nebula Device alebo definovať, vytvárať a nastavovať prvky GUI priamo pomocou vlastného skriptového kódu.

Vlastný kód nepochybne prináša veľkú voľnosť, čo môže byť ale na úkor prehľadnosti a čitateľnosti. Perzistencia objektov je primárne určená pre ukladanie a opätovné načítanie hierarchie objektov z C++ kódu nejakej aplikácie a teda nie je veľmi vhodná pre manuálnu úpravu súborov.

Ukladanie dát pomocou skriptovacieho jazyka vyžaduje volanie C++ funkcií rôznych objektov zo skriptu, pričom samotné dáta sa prenášajú v podobe parametrov týchto funkcií. To môže viesť k nutnosti definovať verejné C++ funkcie i tam, kde to nie je z hľadiska návrhu objektu vhodné. Napríklad môže existovať parameter objektu, ktorý logicky a technicky funguje tým spôsobom, že sa jeho hodnota nastaví iba jeden krát pri vytváraní objektu a ďalšia zmena už nie je možná. Ak by sme takýto parameter chceli nastaviť zo skriptu, musela by existovať i verejná funkcia pre jeho nastavenie. Tá by sa ale tým pádom dala zavolať kedykoľvek. Toto samozrejme nie je veľký problém, no je dobré si to uvedomiť.

Naproti tomu, výhodou skriptu je to, že sa nemusíme zaoberať vlastným systémom pre načítavanie súborov. Vystačíme si so skriptovým rozhraním objektov.

Pre uloženie konfigurácie GUI a šablón bola, vzhľadom na štruktúrovanosť a prehľadnosť, zvolená štandardizovaná technológia XML s využitím knižnice TinyXML. Konfiguračné súbory budú mať nasledovnú štruktúru:

- Každá konfigurácia editora obsahuje jeden hlavný XML dokument. Ten obsahuje zoznam použiteľných ponúk, šablón a dialógov (mimo základných vstavaných dialógov ako je napr. zoznam objektov v scéne).
- Pre každú ponuku, šablónu a dialóg existuje samostatný XML dokument.
- Ovládacie prvky dialógov a šablón sú definované vo vnútri príslušných XML dokumentov.

3.4 Funkčné súčasti konfigurácie

Okrem definície GUI a šablón obsahuje konfigurácia editora aj sadu skriptov, ktorá zabezpečuje fungovanie editora. Ide predovšetkým o funkcie prepojené s ponukami a ovládacími prvkami dialógov a šablón. V C++ kóde editora sa nachádzajú iba nevyhnutné

[6] http://tinyxml.sourceforge.net

služby spojené s editovaním. Takýmto spôsobom je možné rozširovať funkčnosť editora bez zásahu do zdrojového kódu, prípadne upraviť editor aj pre iný typ hry. Skriptovacím jazykom bude podľa zadania Lua. V XML dokumentoch konfigurácie sa budú môcť nachádzať buď cesty ku skriptovým súborom alebo priamo bloky skriptového kódu.

3.5 Základný koncept editora

Editor plne nadviaže na herný engine Mutant. V prvom rade sa jedná o využite tried MUT_Engine a MUT_Game (kapitola 2.2.2). Pravdepodobne bude vytvorený potomok triedy MUT_Game, ktorý ku hre pridá editačné funkcie a ten bude spustený v engine. Takto sa svojím spôsobom vytvorí nový typ „hry“. Nový potomok sa bude starať aj o otvorenie hlavného systémového okna, načítanie konfigurácie editora, spracovanie základného vstupu zo vstupných zariadení a prepínanie pracovného módu (editovanie scény, prototypu, herný mód).

Pre potreby označovania objektov v scéne pomocou myši bude vytvorený špecializovaný server pre zisťovanie objektu pod kurzorom myši (tzv. pick server). Pick server sa stane súčasťou herného enginu tak, aby bol použiteľný nie len pre editor. Pri preštudovaní dokumentácie ku knižnici pre fyzikálnu simuláciu OpenDE (kapitola 2.6) zistíme, že táto knižnica obsahuje vhodné nástroje pre riešenie označovania. Základom pick serveru sa preto stane práca s touto fyzikálnou knižnicou. K editovaným herným objektom či parametrom bude pripojená zvláštna fyzikálna reprezentácia ich povrchu. Povrch rôznych typov objektov bude určený ich tvarom, prípadne sa použije vizuálna trojuholníková sieť viditeľných objektov. Ku neviditeľným objektom (ako napr. zdroj svetla alebo kamera) bude navyše pripojený voliteľný systémový model.

Editor vzniká počas prebiehajúceho vývoja enginu Mutant. I preto je výhodné definovať viacero systémových nastavení pomocou skriptu. Ide predovšetkým o registráciu tried herných objektov, parametrov a systémových modelov. V prípade, že v budúcnosti vznikne v C++ kóde enginu napríklad nový typ parametrov, bude editor schopný bez potreby kompilácie pracovať aj s týmito novými parametrami.

3.6 Dokumentácia

Zdrojový kód editora bude obsahovať komentáre v anglickom jazyku pripravené pre spracovanie systémom Doxygen[7], ktoré spolu so samostatnými dokumentmi (vo formáte *.dox) vytvoria komplexnú programátorskú dokumentáciu. Navyše bude rovnakým spôsobom pripravená i stručná používateľská príručka. Táto príručka sa pomocou DoxyGenu transformuje do komprimovaného súboru User.chm, ktorý bude možné otvoriť priamo z hlavnej ponuky editora. Viac informácií o programe Doxygen je možné nájsť v manuáli [8].

[7] http://www.stack.nl/~dimitri/doxygen

4 Implementácia riešenia

4.1 Rozdelenie súborov

Umiestnenie zdrojových a dátových súborov je nasledujúce:

- MutantProgram\Editor\Build – projektové súbory pre MS Visual Studio 2003 .NET spolu so systémovými zdrojovými súbormi a obrázkami
- MutantProgram\Editor\Classes – zdojové súbory editora. Každá trieda má svoj hlavičkový súbor *názov*.h a zdrojový súbor *názov*.cpp. V prípade, že trieda obsahuje skriptový interface, je uložený v súbore *názov*_cmds.h.
- MutantProgram\Editor\Classes\Helper – pomocné triedy.
- MutantProgram\Editor\Doc – súbory dokumentácie (*.dox).
- MutantProgram\Editor\Documents\Doxygen – konfiguračné súbory pre program Doxygen.
- MutantProgram\Editor\Documents\Sources – tu je uložená programátorská dokumentácia vygenerovaná programom Doxygen.
- MutantProgram\Editor\Documents\User – miesto uloženia používateľskej dokumentácie.

- MutantProgram\Engine\Build – obsahuje projektové súbory enginu Mutant.
- MutantProgram\Engine\Classes – zdrojové súbory enginu.
- MutantProgram\Engine\Doc – súbory dokumentácie.

- MutantProgram\Documents\Sources – programátorská dokumentácia enginu.
- MutantProgram\Foundation – podporné knižnce enginu.
- MutantProgram\Nebula2 – zdrojové súbory knižnice Nebula Device.
- MutantProgram\ODE – knižnica OpenDE.

- MutantGame\Bin – obsahuje spustiteľnú verziu editora.
- MutantGame\Editor – dátové súbory editora.
- MutantGame* – ďalšie adresáre majú intuitívne názvy a obsahujú ukážkové levely, prototypy a ostatné dátové zdroje.

4.2 Jadro editora

Základom editora je aplikačná trieda MUT_EditorApp odvodená od CWinApp. Keďže Nebula Device obsahuje vlastný cyklus pre spracovanie správ systému Windows, je potlačená funkcia Run. Táto funkcie vytvára a spúšťa engine Mutant s prenastaveným typom hry. Novým „typom hry“ sa stáva potomok triedy MUT_Game nazvaný EDT_Editor. Je uložený v súbore MutantGame\System\editor.n2, kde sa nachádzajú aj jeho základné nastavenia. (Pre porovnanie viď kapitolu 2.2.2.)

EDT_Editor počas otvárania v prvom rade vytvára pomocou MFC hlavné okno aplikácie so stavovým panelom (status bar) a panelom pre zobrazovanie postupu (progress bar). V klientskej oblasti hlavného okna je navyše vytvorený panel určený pre výstup enginu Nebula.

Ďalej je načítaná zvolená konfigurácia editora (MutantGame\Editor\mutant.xml), čo zo sebou prináša vytvorenie objektov pre každú ponuku, šablónu a dialóg. V prípade ponuky sa v MFC dopredu vytvorí funkčný objekt. Šablóny a dialógy sú iba načítané a predpripravené. Funkčné dialógy a ovládacie prvky šablón sa vytvárajú až pri použití. Popis povoleného obsahu XML dokumentov sa nachádza v používateľskej príručke (príloha C).

Oproti triede MUT_Game je navyše spustený ešte vlastný konfiguračný skript (MutantGame\Scripts\System\setup_editor.lua). Ten definuje viacero globálnych funkcií predovšetkým pre editovanie herných objektov a prototypov (kapitola 4.6), vytvára vstavané dialógy (kapitola 4.3.3) a registruje systémové modely (kapitola 4.7.1).

4.3 Základné triedy

4.3.1 Ponuky

Hlavú *ponuku* (*menu*) reprezentuje trieda EDT_Menu. Tá pri načítaní ponuky zo XML dokumentu vytvorí pomocou MFC kompletnú ponuku i s vnorenými položkami. Táto ponuka je neviditeľná až pokým sa nepriradí k hlavnému oknu aplikácie. To zabezpečuje funkcia setMenu triedy EDT_Editor. Takýmto spôsobom môže v pamäti existovať niekoľko ponúk súčasne, pričom sú zobrazované podľa potreby. Editor používa samostatnú ponuku pre každý pracovný mód. Po spustení je zobrazená úvodná ponuka s identifikátorom „default“, ak existuje. Ostatné ponuky sa zobrazujú so zmenou pracovného módu zo skriptu.

Položky ponuky môžu mať definovaný skript alebo cestu k súboru so skriptom, ktorý sa spustí pri zvolení danej položky.

Ponuka môže definovať k jednotlivým položkám *klávesové skratky* (*hot keys*) ako napríklad Ctrl+N alebo Ctrl+Shift+O. V klasickej aplikácii typu MFC sú klávesové skratky ponúk registrované a spracovávané automaticky. Editor však používa predefinovaný cyklus správ, ktorý sa nachádza v engine Nebula Device, preto je potrebné obhospodáriť klávesové skratky ručne. To si vyžaduje vytvorenie špeciálnej tabuľky akcelerátorov pomocou Windows API a jej nastavenie v engine. Ten sa stará o spracovanie tabuľky prostredníctvom triedy nWin32WindowHandler. Trieda však spočiatku nemala možnosť nastavenia vlastnej tabuľky akcelerátorov. Preto jej bola táto služba pridaná spolu so zverejnením objektu triedy nWin32WindowHandler tak, aby bolo možné službu použiť aj z kódu aplikácie (mimo samotný engine).

4.3.2 Dialógy

Editor rozlišuje dva typy dialógov. Sú to tzv. *vstavané* a *používateľské dialógy*. Obidva typy však majú rovnaký základ, triedu EDT_DialogBase. Táto trieda poskytuje jednotný prístup k obom typom dialógov: zobrazenie, modálne zobrazenie, skrytie, nastavenie pozície a veľkosti. Virtuálne funkcie createDialog a createDialogModal je potrebné vo všetkých odvodených triedach potlačiť pre vytvorenie konkrétneho typu dialógu.

Používateľské dialógy reprezentuje trieda EDT_Dialog. Pri načítavaní zo XML dokumentu sa pomocou Windows API vytvorí a naplní *šablóna dialógu* (štruktúra

DLGTEMPLATE). Z tejto šablóny je potom možné vytvoriť a zobraziť inštanciu samotného dialógu. V prípade, že je dialóg zobrazený nemodálne, vytvorí sa jeho inštancia iba raz pri prvom použití. Pri ďalšom skrývaní a zobrazovaní ostáva dialóg v pamäti. V prípade modálneho dialógu, sa jeho inštancia pri zobrazení vytvorí a pri zavretí ihneď automaticky zmaže (takto fungujú modálne dialógy v MFC všeobecne).

4.3.3 Vstavané dialógy

Editor obsahuje šesť vstavaných dialógov (*common dialogs*). Ich triedy majú predponu EDT_CD_. Vzhľad a ovládacie prvky dialógov sú definované v súbore zdrojov aplikácie (viď kapitola 2.3.5). Rovnako tak funkčnosť vstavaných diaógov je do veľkej miery zabezpečená priamo v C++ kóde.

- Dialóg „O programe“ (EDT_CD_AboutBox) zobrazuje okno s informáciami o programe.

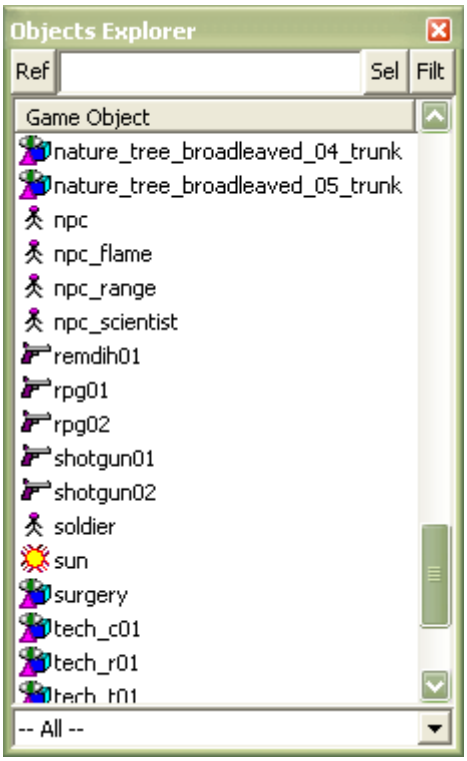
- Dialóg pre otvorenie/uloženie súboru (EDT_CD_FileDialog) otvorí štandardné okno pre otvorenie či uloženie súboru tak ako je známe z iných aplikácií. Dialóg môže mať nastavený úvodný adresár, filter pre prípony súborov, meno súboru a predvolenú príponu súboru. Špeciálnou vlastnosťou je, že dialóg spolupracuje s cestami adresárov zadanými pomocou skratiek (viď kapitola 2.1.5). Ak úvodný adresár obsahuje skratku a je vybraný súbor z tohto adresára alebo jeho podadresára, tak výsledná cesta k súboru rovnako obsahuje túto skratku. K tomuto účelu bola enginu Nebula Device pridaná služba pre skracovanie ciest. Táto služba pri zadaní cesty a skratky vráti, v prípade že cesta obsahuje časť zhodnú so skratkou, cestu zapísanú pomocou skratky; inak vráti cestu bez zmeny.

- Výpis herných objektov v otvorenej scéne sa nachádza v dialógu nazvanom „Objects Explorer“ (EDT_CD_ObjectsExplorer, Obrázok 4.1). Okrem výpisu tu nájdeme editačné pole pre filtrovanie objektov podľa názvu a filter pre rôzne typy objektov. Ich použitie je vysvetlené v používateľskej príručke (príloha C). Označením položky v zozname dôjde k zvoleniu daného herného objektu. Opačne, zvolenie herného objektu v scéne spôsobí označenie príslušnej položky. Editor pritom dovoľuje označiť aj viacero objektov naraz. Jednotlivé typy herných objektov sú v zozname odlíšené ikonou. Typ herného objektu spolu s jeho ikonou je možné registrovať funkciou addObjectClass.

- Pre zobrazenie stromovej hierarchie otvoreného prototypu slúži dialóg nazvaný „Prototype View“ (EDT_CD_PrototypeView, Obrázok 4.2). Dialóg obsahuje aj panel nástrojov s tlačidlami pre každý typ parametrov. Funkčnosť tohto panelu nástrojov je značne previazaná so skriptovými funkciami (viď kapitolu 4.6). Označením položky v stromovej hierarchii dôjde k zvoleniu príslušných parametrov a opačne. Nie je možné označiť viacero parametrov naraz. Jednotlivé typy parametrov sú opäť od seba odlíšené ikonou. Typy parametrov sa registrujú pomocou funkcie addParamClass.

- Ako cieľový dialóg pre zobrazenie ovládacích prvkov šablón sa používa dialóg Properties (EDT_CD_Properties). Dialóg sám osebe obsahuje iba tlačidlo pre aplikovanie zmien v ovládacích prvkoch.

- Posledným vstavaným dialógom je dialóg pre editovanie hodnôt jedného riadku ovládacieho prvku typu tabuľka (EDT_CD_TableDialog). Princíp fungovania je nasledovný: ak šablóna obsahuje ovládací prvok typu tabuľka, v dialógu Properties sa zobrazí jednoduchá textová tabuľka. Stlačenie tlačidla Insert na klávesnici vyvolá pridanie riadku do tabuľky. Zobrazí sa tabuľkový dialóg s ovládacími prvkami pre nastavenie hodnôt – jeden prvok pre každý stĺpec tabuľky. Typ ovládacích prvkov je daný typom odpovedajúcich stĺpcov tabuľky.



Obrázok 4.1: Dialóg Objects Explorer

Obrázok 4.2: Dialóg Prototype View

4.3.4 Ovládacie prvky

Dialóg môže v sebe obsahovať *ovládacie prvky* (*controls*). Keďže cieľom konfigurácie editora nie komplexná definícia grafického používateľského rozhrania, sú ovládacie prvky používané v editore špecifické. Väčšina z nich je určená pre zobrazenie a nastavenie hodnoty určitého typu ako je napr. číslo, text, súbor, trojzložkový vektor, alebo tabuľka. Jeden ovládací prvok sa teda môže skladať z viacerých komponent. V prípade prvku pre nastavenia vektora to sú tri editačné polia vedľa seba a pod. Na výber máme aj ďalšie užitočné typy prvkov: tlačidlo, textový nápis a textový nápis pre zobrazenie ľubovoľnej hodnoty (textu, čísla, vektoru, ...). Ďalším zjednodušením je fakt, že ovládacím prvkom nie je potrebné definovať pozície a rozmery. Ovládacie prvky budú mať v takomto prípade rovnakú štandardnú veľkosť a budú v dialógu umiestnené pod sebou.

Triedy pre jednotlivé typy ovládacích prvkov sú odvodené od spoločného predka s názvom EDT_CTL_Control a majú predponu EDT_CTL_. Tieto triedy sú spoločné pre použitie v dialógoch ako aj v šablónach parametrov a herných objektov. Ich špeciálnou vlastnosťou je preto možnosť prepojenia s parametrami prototypu alebo s herným objektom. Týmto prepojením je jednak textový identifikátor cieľovej vlastnosti C++ objektu, a potom sada skriptov pre prenos hodnôt medzi objektmi a ovládacími prvkami.

Navyše je možné ku každému prvku definovať skripty pre otestovanie správnosti hodnoty a pre reakciu na zmenu hodnoty. Špeciálnymi skriptami sú potom reakcie na vizuálne editovanie objektu. Napríklad pri posúvaní herného objektu v scéne sa takýmto skriptom prenesie jeho aktuálna poloha do ovládacieho prvku pre nastavenie pozície. Bližší popis editačných funkcií sa nachádza v kapitole 4.6.

Všetky ovládacie prvky dialógu alebo šablóny zastrešuje trieda EDT_Controls. Tá umožňuje jednak vytvorenie ovládacích prvkov v zadanom dialógu, no jej hlavý prínos je pri hľadaní prieniku šablón. Nájsť prienik šablón znamená nájsť všetky ovládacie prvky, ktoré majú v každej šablóne rovnaký identifikátor cieľovej vlastnosti. To je potrebné v prípade, že je označených naraz niekoľko herných objektov. V dialógu s nastaviteľnými vlastnosťami sú potom zobrazené iba ovládacie prvky spoločných vlastností všetkých objektov.

4.3.5 Šablóny

Šablónu herného objektu alebo parametrov reprezentuje trieda EDT_Template. Tá používa pre prácu s ovládacími prvkami spomínanú triedu EDT_Controls. Pre každý typ herného objektu a parametrov existuje samostatná šablóna. Po označení objektu či parametrov sa aktivuje príslušná šablóna a tá vytvorí v zadanom dialógu svoje ovládacie prvky. Šablóna obsahuje počítadlo, ktoré určuje, koľko krát bola aktivovaná. V prípade, že je označených viacero herných objektov naraz, aktivuje sa viacero šablón. Posledne aktivovaná šablóna nájde prienik ovládacích prvkov všetkých aktívnych šablón a vytvorí tieto prvky v zadanom dialógu.

4.4 Pomocné triedy

Dialógy poskytované knižnicou MFC majú značne obmedzené možnosti použitia. Neráta sa tu s tým, že by programátor mohol potrebovať napríklad dialóg s panelom nástrojov alebo dialóg v ktorom by automaticky fungovali posuvníky. Dokonca pri nemodálnych dialógoch nie je zabezpečené ani preskakovanie medzi ovládacími prvkami pomocou klávesu Tab (resp. Shift+Tab). Preto sa súčasťou editora stali aj pomocné triedy, ktoré prinášajú chýbajúcu funkčnosť.

- Prvou z pomocných tried je CToolbarDialog odvodená od CDialog. Ako napovedá názov, úlohou tohto dialógu je zobrazenie a zaobstaranie funkčnosti panelu nástrojov vo vnútri dialógu. Panel je fixný a je možné ho umiestniť k ľubovoľnému okraju dialógu.

- Ďalšou triedou je CScrollDialog. Trieda zabezpečuje automatické fungovanie posuvníkov. Jediné, čo je potrebné, je nastaviť rozmery oblasti, ktorá ma byť posúvaná (funkcia SetScrollArea). Dialóg podporuje aj vertikálne posúvanie pomocou kolieska myši.

- Trieda CDialogMessageHook sa stará o správne spracovanie klávesy Tab a ostatných klávesových skratiek v nemodálnych dialógoch. Jej použitie je veľmi jednoduché. Statická funkcia InstallHook pridá funkčnosť k zadanému dialógu, funkcia UninstallHook zase odoberie. Princíp jej fungovania spočíva v tom, že všetky správy poslané operačným systémom hlavnému oknu preposiela registrovaným dialógom.

4.5 Pracovné módy

Editor podporuje tri pracovné módy: editácia scény, editácia prototypu a herný mód.

Pri otvorení levelu je nastavený mód editácie scény. Samotný level je načítaný pomocou enginu Mutant (zatiaľ nie sú vytvorené žiadne herné objekty). Následne je tento level uložený do pomocného súboru a odstránený z pamäte. Ďalej sa načíta a spracováva už len tento pomocný level. K levelu sú pridané dva systémové objekty: kamera a svetlo. Následne je level aktivovaný s použitím systémovej kamery (vytvoria sa herné objekty) a jeho aktualizácie sú pozastavené – level „zamrzne“. Poslednou pridanou súčasťou levelu je priestor pre označovanie objektov pomocou myši.

Dôvodom, prečo sa používa pomocný level je podpora pre herný mód. Pri prechode do herného módu z editácie scény sa pomocný level uloží do originálneho súboru (bez systémových objektov), potom je herný level načítaný a spustený bez ďalších zásahov. Pomocný level so systémovými objektmi a priestorom pre označovanie objektov je stále v pamäti. Pri ukončení herného módu sa jednoducho zavrie herný level a aktivuje pomocný. Bez pomocného levelu by bol prechod do herného módu a späť buď príliš komplikovaný alebo príliš pomalý. Takto sa bez problémov zachová priestor pre označovanie, systémové objekty, pomocné vizualizácie, poloha kamery, označenie objektov a pod.

Pre editáciu prototypu sa používa opäť špeciálny pomocný level. Ten obsahuje na začiatku iba systémovú kameru a svetlo. Pri aktivovaní prototypu sa pridá do scény herný objekt (resp. objekty) vytvorený pomocou daného prototypu. Nakoniec je pridaný priestor pre označovanie vizualizovaných parametrov prototypu pomocou myši.

4.6 Editačné funkcie

Základné editačné služby ponúka objekt triedy EDT_Editor. Pri editácii prototypu sa jedná o označenie parametrov, pridávanie parametrov do prototypu či kontajnera, ich odoberanie a aplikovanie nových hodnôt parametrov zo šablóny. Pri editácii scény zase o importovanie nových herných objektov do scény a aplikovanie zmien v šablóne. Ďalej sa tu dajú nájsť aj pomocné služby: vytvorenie priestoru pre označovanie objektov pomocou myši, zmena identifikátorov aktuálneho módu a uloženia scény/prototypu, zmena aktuálnej ponuky, aktivovanie prototypu a parametrov a pod. Editor zabezpečuje aj vizuálne editovanie – reakciu na rôzne klávesové skratky v kombinácii s pohybom myši.

Všetky tieto služby sú na nízkej úrovni a vykonávajú iba nevyhnutnú časť práce, ktorá bola vhodná pre implementovanie v C++ kóde. Nenachádzajú sa v nich hlbšie kontroly alebo testovanie rôznych závislostí. Veľká časť editačných funkcií je implementovaná v skripte (súbory editobjects.inc.lua a editparameters.inc.lua v MutantGame\Editor\Scripts\Inc). Tieto funkcie slúžia predovšetkým pre prenos hodnôt medzi ovládacími prvkami šablón a C++ objektmi a pre komplexnú manipuláciu s parametrami.

4.7 Vizualizácia objektov

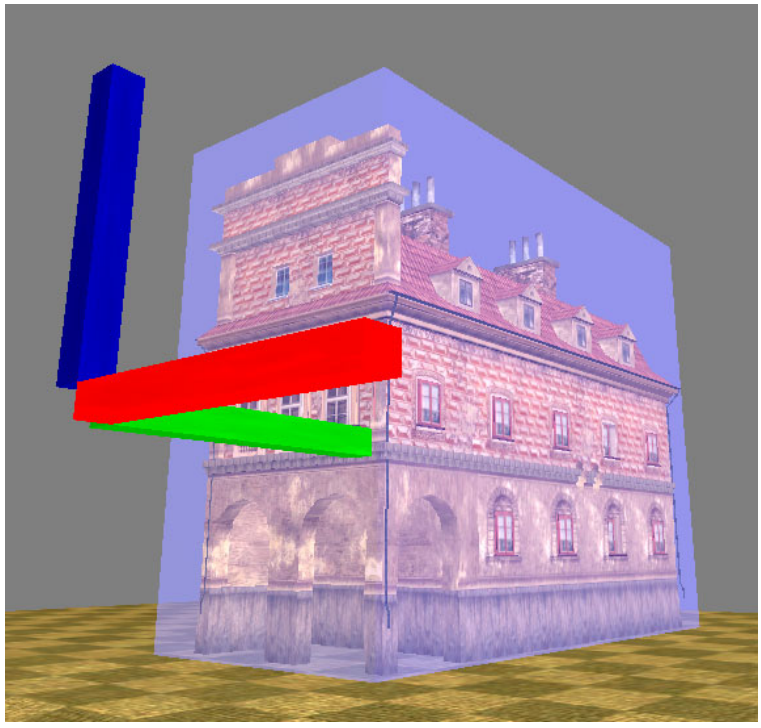
4.7.1 Systémové modely

Pri editovaní scény alebo prototypu je potrebné vidieť aj objekty, ktoré sú inak neviditeľné. Takými sú napríklad kamery alebo zdroje svetla. K objektom (respektíve ich C++ triedam) je možné registrovať tzv. systémové modely. Najlepším miestom pre túto registráciu je konfiguračný skript setup_editor.lua. V prípade herných objektov je možné použiť špeciálny názov triedy „godefault“. Systémový model priradený ku godefault bude použitý v prípade každého herného objektu, ktorý je neviditeľný a nemá nastavený svoj vlastný systémový model.

4.7.2 Pomocné objekty

Mnohé herné objekty dokážu zobraziť aj ďalšie pomocné útvary a texty, ktoré slúžia pre ladenie. Túto funkčnosť zabezpečuje už engine Mutant a editor ju plne využíva. Na výber máme: systémové informácie, kubické obálky (bounded boxes), kolízne obálky, kvádre pre optimalizáciu vykresľovania (occluders), navigačné siete a RPG údaje o herných objektoch (život, počet nábojov a pod).

Editor k tomu ešte pridáva ladiace útvary objektov slúžiacich pre označovanie myšou (pick objects). Táto funkcia je ale prístupná iba pri ladiacom zostavení aplikácie.



Obrázok 4.3: Ukážka systémového modelu kamery
a vizualizácie kubickej obálky.

4.8 Picking

Dôležitou funkciou editora je označovanie objektov pomocou myši priamo v zobrazenej scéne (*picking*). Pre tento účel bol vytvorený tzv. pick server spolu s pick priestorom a agentom (triedy MUT_PCK_Server, MUT_PCK_Space a MUT_PCK_Agent). Tieto triedy implementujú picking pomocou knižnice OpenDE (kapitola 2.6).

Počas pridávania herného objektu do pick priestoru sa vytvorí pre tento objekt pick agent a sada geometrických útvarov reprezentujúcich objekt v pick priestore. Tieto útvary si môže každý typ herného objektu vytvoriť sám prostredníctvom virtuálnej funkcie addPickGeom. Aktuálne existuje pre všetky typy herných objektov jedna spoločná funkcia, ktorá robí nasledovné: v prípade, že objekt obsahuje viditeľnú zložku, prevediú sa vizuálne trojuholníkové siete všetkých zobraziteľných uzlov objektu (viď kapitolu 2.1.2) na kolízne siete používané v ODE. V opačnom prípade je požiadaný o vytvorenie geometrických útvarov prototyp daného herného objektu. Prototyp sa postupne snaží vytvoriť útvary pre jednotlivé parametre v tomto poradí: vizuálne parametre, kolízne, kamerové, svetelné. Akonáhle sa mu to podarí, ďalej nepokračuje.

Analogicky funguje aj pridávanie parametrov prototypu do pick priestoru. Najprv je vytvorený pick agent a následne zavolaná virtuálna funkcia addPickGeom daných parametrov. Pick agent teda môže byť naviazaný ako na herný objekt tak i na parametre. Pri pohybe herného objektu v scéne sa prostredníctvom agenta aktualizuje aj poloha príslušných geometrických útvarov.

Pick agent podporuje rôzne typy geometrických útvarov: kvádre, gule, kapsuly, a trojuholníkové siete. Pre vytvorenie kolíznej trojuholníkovej siete je použitá funkcia NewTriMesh triedy nOpendeServer. Táto funkcia vytvára trojuholníkovú sieť pomocou zavádzača (*loader*), ktorý za normálnych okolností načítava sieť zo súboru. Za účelom konverzie vizuálnej siete (trieda nMesh2) na kolíznu sieť v ODE (nOpendeTriMesh) bol vytvorený špeciálny zavádzač nMemoryMeshLoader, ktorý „načítava“ sieť priamo z pamäte. Keďže vizuálne siete už boli raz načítané spolu s herným objektom, tento trik značne urýchli vytváranie kolíznych sietí. Namiesto cesty k súboru je zadaný jednoducho smerník k vizuálnej sieti. Trieda nMemoryMeshLoader sa stala súčasťou enginu Nebula Device.

4.9 Dokumentácia

V rozbore riešenia bolo načrtnuté, že súčasťou editora bude používateľská príručka vo formáte CHM. Túto príručku je možné vygenerovať aplikáciou Doxygen v spolupráci s programom HTML Help Workshop, ktorý je súčasťou operačného systému Windows (C:/Program Files/HTML Help Workshop/hhc.exe). Konfiguračný súbor user.cfg pre používateľskú príručku sa nachádza v adresári MutantProgram\Editor\Documents\Doxygen. Výsledná dokumentácia vo formáte CHM je uložená na adrese MutantGame\Editor\Documents\User.chm. Navyše je prístupná i v HTML formáte na tomto mieste: MutantProgram\Editor\Documents\User\html\index.html.

Aby bolo možné spustiť používateľskú príručku z ponuky aplikácie (teda zo skriptu), obsahuje trieda MUT_Editor metódu showHelp(), ktorá využíva knižnicu htmlhelp.lib.

Vytvorený je i konfiguračný súbor sources.cfg pre generovanie programátorskej dokumentácie. Tá sa v HTML podobe ukladá do adresára MutantProgram\Editor\Documents\Sources\html. Dokumentácia editora je tiež súčasťou programátorskej dokumentácie enginu Mutant.

4.10 Problémy pri implementácii

Implementácia editora podľa rozboru priniesla celý rad skrytých problémov. Jednalo sa hlavne o problémy spojené s integráciou riešenia do existujúceho prostredia enginu Nebula Device.

4.10.1 MFC vs. Nebula Device

Ako prvé sa objavili problémy s prepojením MFC a Nebuly. Nebula totiž kvôli ladeniu a kontrole pamäti používa predefinované operátory new a delete. To isté však robí aj MFC, čo spôsobuje komplikácie pri linkovaní programu. Pre potreby editora sú vhodnejšie operátory z Nebuly (aby fungovanie editora bolo identické s fungovaním hry). Keďže sa jednotlivé statické knižnice z Nebuly linkujú skôr ako MFC, bol použitý prepínač linkovania /FORCE, ktorý umožnil linkovanie programu napriek dvojitej definícii operátorov. Druhá definícia (z MFC) sa pritom ignoruje.

Nebula obsahuje tiež automatické fungovanie klávesovej skratky Alt+Enter, ktorá má za úlohu prepínanie medzi zobrazovaním v okne a na celej obrazovke. Táto funkcia však pri kombinácii so zobrazovaním vo vnútri rodičovského okna zlyhá. Preto bolo použitie klávesovej skratky v editore zakázané.

4.10.2 STL vs. Nebula Device

Pri implementácii nielen editora ale i enginu Mutant boli vo veľkej miere využívané triedy z STL (Standard Template Library) ako sú spojové zoznamy, vektory a pod. Editor však postupne začal vykazovať veľkú nestabilitu, predovšetkým pri rýchlych a plynulých operáciách, napr. pri vizuálnom posúvaní objektu v scéne. Táto nestabilita sa však prejavovala iba v prípade, keď bola aplikácia preložená s optimalizáciami a bez ladiacich informácií. Preto bolo hľadanie zdroja padania programu problematické. Nakoniec sa ukázalo, že aplikácia zhavaruje práve pri používaní STL. Akonáhle boli triedy STL nahradené za nejakú alternatívu (napr. polia a spojové zoznamy implementované v Nebule), nestabilita zanikla. Preto došlo ku globálnemu nahradeniu tried z STL v editore i v celom engine Mutant.

4.10.3 Použitie enginu

Viaceré menšie problémy prinieslo nesprávne použitie funkcií enginu. Napríklad pre prácu s myšou je používaná tzv. relmouse (relatívna myš), ktorá vykazuje pohyb aj v prípade, že sa nachádza mimo výstupného okna, či na okraji obrazovky (kapitola 2.1.4). To je potrebné pre vizuálne editovanie polohy a otočenia objektov. Toto virtuálne vstupné zariadenie však mimo výstupného okna vykazuje i stlačenie tlačidla. Dokonca i keď je výstupné okno neaktívne a kurzor myši sa nachádza nad niektorým dialógom alebo keď je aplikácia minimalizovaná. Tým pádom dochádzalo ku veľmi kurióznym situáciám. Riešenie bolo v skutočnosti veľmi jednoduché: pre generovanie udalostí o tlačidlách myši

stačilo použiť iné virtuálne zariadenie – „obyčajnú myš“ – ktoré funguje tak ako by sme očakávali.

Iným typom komplikácie bolo padanie editoru pri jeho vypínaní. Ku výnimke dochádzalo v deštruktore triedy pre prácu so signálmi. V objekte tejto triedy sa totiž nachádzal alokovaný objekt signálu, ktorý tam už nemal byť. V samotnej hre, tento problém nebol. Ako sa ukázalo, chyba bola v statickom deklarovaní triedy MUT_Engine. Po úprave na dynamickú alokáciu sa upravilo poradie volania deštruktorov a problém bol vyriešený.

4.10.4 Chyby v engine Nebula Device

Pri implementácií editora bolo objavených niekoľko chýb priamo v engine Nebula Device.

- Prvá chyba enginu sa objavila hneď na začiatku implementácie editora. Nebula totiž odmietala otvoriť dcérske okno vo vnútri hlavného okna vytvoreného pomocou MFC. Táto chyba bola opravená a nahlásená pod číslom 216 v systéme pre hlásenie chýb enginu[8].

- Pri vytváraní zavádzača nMemoryMeshLoader (viď kapitola 4.8) sa našli chyby vo funkciách LoadN3d2 a LoadNvx2 triedy nOpendeTriMesh. Chyba je registrovaná pod číslom 303.

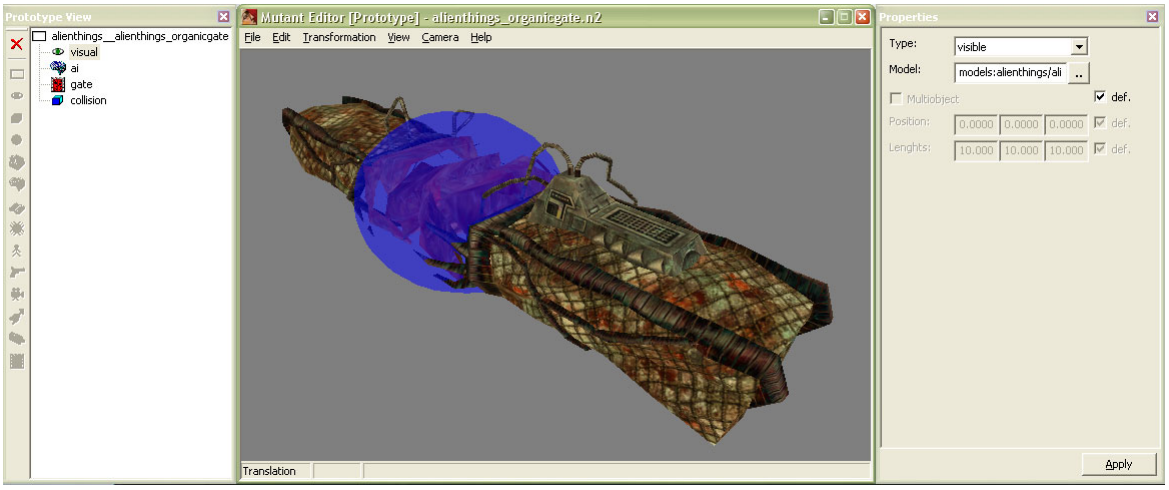
- Posledná chyba bolo nájdená v deštruktore triedy nD3D9Server a má číslo 352.

[8] http://nebuladevice.cubik.org/bugs

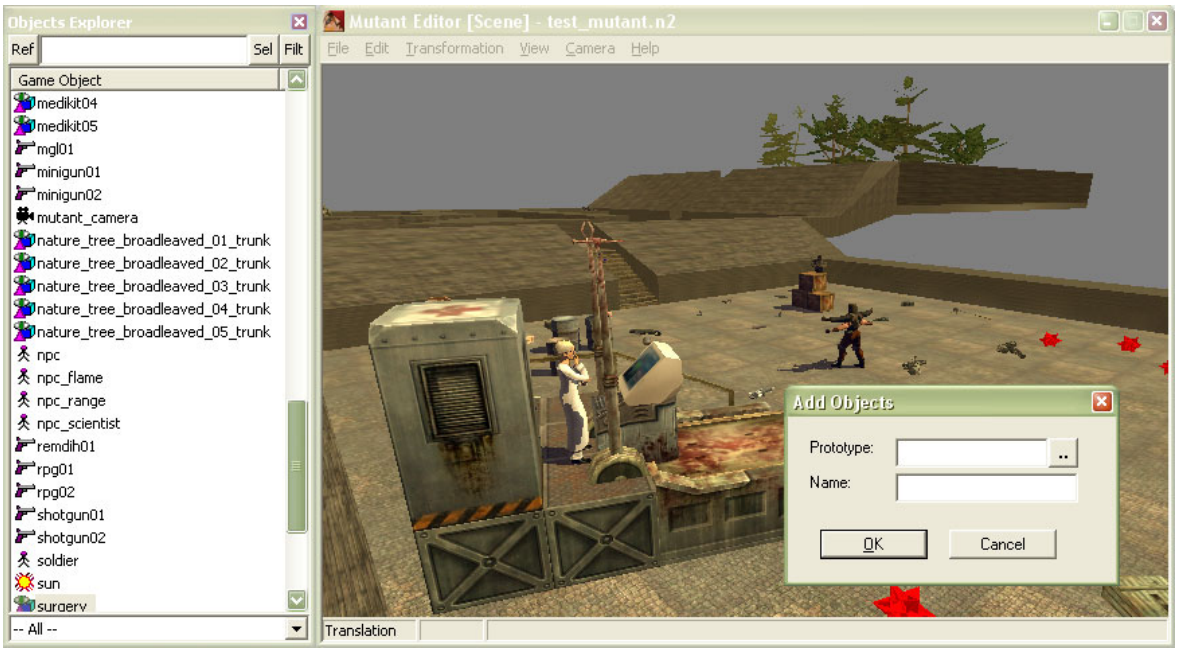
5 Využitie aplikácie

Aplikácia bola vytvorená na objednávku filmy Napoleon Games, ktorá editor využíva pri vytváraní pripravovanej 3D akčnej hry. Editor však má, vďaka voliteľnej konfigurácii, potenciál byť ľahko upravený a použitý aj pre ďalšie hry založené na engine Mutant.

5.1 Ukážky z aplikácie



Obrázok 5.1: Ukážka z editácie prototypu



Obrázok 5.2: Ukážka z editácie scény

5.2 Testovanie implementácie

Editor bol testovaný priebežne počas vývoja nielen na skúšobných objektoch a scénach ale aj priamo na budúcom leveli hry s desiatkami herných objektov a takmer tisícom povrchových sietí. A práve tieto siete najviac ovplyvňujú rýchlosť otvorenia scény v editore. Počas otvárania najprv prebieha načítavanie levelu enginom a potom sa pre všetky objekty vytvárajú zástupcovia v priestore pre označovanie objektov. V prípade vizuálnych herných objektov, ktorých povrch je definovaný sieťou trojuholníkov (také sú takmer všetky), sa príslušné vizuálne siete prevádzajú do fyzikálnej reprezentácie.

V ranných štádiách vývoja pickingu (ako sa označovanie pomocou myši zvykne nazývať) boli najprv vizuálne siete ukladané do pomocného súboru a následné načítané štandardnou službou fyzikálneho servera. Dôvod, prečo nebolo možné načítať fyzikálne siete rovno zo súborov vizuálnych sietí, bol ten, že sieť definovaná v súbore musí pre vrcholy definovať iba jednu zložku – pozíciu vrcholu. Inak ju fyzikálny server odmietne. Pred uložením siete do pomocného súboru sa všetky ďalšie zložky vrcholov odstránili. Prevod jednej siete si teda vyžadoval nasledujúce kroky:

- Skopírovanie vizuálnej siete v pamäti
- Odstránenie nadbytočných zložiek
- Uloženie siete do súboru
- Odhodenie kópie siete
- Načítanie siete zo súboru priamo fyzikálnym serverom

To sa samozrejme veľmi rýchlo ukázalo ako príliš pomalé a prakticky nepoužiteľné. Z toho dôvodu vznikol špeciálny zavádzač, ktorý načítava fyzikálne siete priamo z vizuálnych, uložených v pamäti (viď kapitola 4.8). Použitie zavádzača zásadným spôsobom urýchlilo otváranie veľkých scén (na úroveň použiteľnú bez problémov v praxi).

Z pomedzi ďalších zvolených techník je ešte dobré pripomenúť použitie knižnice TinyXML pre spracovanie XML dokumentov konfigurácie editora. Táto knižnica je veľmi jednoduchá a poskytuje len základnú funkčnosť. A práve v tom je jej výhoda. Načítanie konfigurácie a teda spustenie aplikácie je o to rýchlejšie. Síce tu nie je možnosť kontrolovať správnosť štruktúry dokumentu voči nejakej definícii, ale to ani nie je potrebné. Bežný používateľ s konfiguračnými súbormi editora vôbec nepracuje. Tie má na starosti poskytovateľ aplikácie.

Prácu s dialógmi aplikácie urýchľuje technika pri ktorej sa dialóg vytvára až pri prvom použití. V prípade nemodálnych dialógov potom ostáva dialóg v pamäti aj po jeho uzatvorení. Pri ďalšom použití sa už iba znovu zobrazí.

Súčasťou aplikácie sú aj ukážkové prototypy herných objektov a levely. Prototypy sa nachádzajú v adresári MutantGame\Prototypes, levely v MutantGame\Levels. Niektoré levely, pri spustení v hernom móde, podporujú prepnutie do testovacieho režimu (kláves F11), v ktorom často využívajú klávesy 1, 2, 3 atď. pre rôzne testovacie účely enginu. Napríklad v leveli test_physics sa klávesom 1 zhadzujú kocky, klávesom 2 vrhajú gule, klávesy 3 a 4 zamŕzajú/odmŕzajú level a kláves 0 zmaže nové objekty. Kláves Space prepína pohyb kamery.

5.3 Ďalší rozvoj programu

Existujú tri hlavné smery pre ďalší rozvoj programu.

- Prvým je pridávanie nových editačných funkcií, ktoré by uľahčovali prácu dizajnérovi. Takou by mohla byť napríklad funkcia pre automatické vertikálne uloženie zvolených herných objektov na terén a pod.

- Druhým sú ďalšie pomocné vizualizácie a vizuálne editovanie pomocou myši bez použitia klávesnice. Tu patrí napríklad zvýraznenie označených herných objektov v scéne, naznačenie súradnicových osí spojených s objektom, zmena pozície objektu ťahaním za súradnicovú os atď.

- Posledným smerom rozvoja sú zmeny odvodené od zmien v engine Mutant. Jedná sa predovšetkým o pridávanie nových typov herných objektov a prototypov, a pridávanie nových vlastností objektov, parametrov a levelov.

6 Záver

Cieľom tejto diplomovej práce bolo vytvorenie kompletnej a funkčnej aplikácie – editora objektov a scén pre hru Mutant. Aplikácia pritom plne využíva prostredie vizualizačného enginu Nebula Device a herného enginu Mutant. Zadanie práce sa s postupným vývojom enginu mierne upravilo a doplnilo. Zmeny sa týkali predovšetkým spôsobu implementácie prototypov a manipulácie s nimi. Aplikácia bola úspešne dokončená a odsúhlasená zadávateľom.

Program predstavuje základnú verziu editora, ktorú je možné ďalej rozširovať a dopĺňať i bez zásahu do zdrojového kódu. To je možné vďaka voliteľnej konfigurácii definovanej pomocou XML dokumentov a skriptov v jazyku Lua. Konfigurácia editora definuje jednoduché grafické používateľské rozhranie. Na výber sú ponuky, dialógy s ovládacími prvkami a špecializované šablóny pre úpravu parametrov zvoleného objektu. Jednotlivé prvky používateľského rozhrania sú vytvárané pomocou technológie MFC a majú teda štandardnú podobu známu zo systému Windows. Funkčnosť ovládacích prvkov je prepojená výhradne zo skriptovými funkciami, ktoré potom môžu využívať služby editora alebo enginu.

Literatúra

[1] R. Smith. *Open Dynamics Engine v0.5 User Guide*. 2006. http://opende.sourceforge.net

[2] R. Ierusalimschy, L. H. de Figueiredo, W. Celes. *Lua 5.0 Reference Manual*. Tecgraf, 2003. http://www.lua.org

[3] J. Kosek. *XML pro každého, podrobný průvodce*. Grada, 2000.

[4] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau. *Extensible Markup Language (XML) 1.0 (Third Edition)*. W3C, 2004. http://www.w3.org/TR/REC-xml.

[5] I. Mlýnková. *XML Schema a jeho implementace v prostředí relační databáze*. 2003.

[6] Kolektív autorov. *The Nebula Device 2 Documentation*. 1999-2006. http://nebuladevice.cubik.org/

[7] J. Rohlík. Mutant engine & authoring tool. Napoleon Games, 2006.

[8] D. van Heesch. *Doxygen Manual*. 1997-2006. http://www.stack.nl/~dimitri/doxygen

[9] J. Prosise. *Programování ve Windows pomocí MFC*. Computer Press, 2002.

A Obsah CD-ROM

Súčasťou tejto diplomovej práce je i priložený CD-ROM obsahujúci jednak text práce, no predovšetkým inštalačný program aplikácie. Súčasťou inštalácie sú aj zdrojové súbory editora a ukážkové dátové súbory – prototypy a scény spolu s potrebnými modelmi, textúrami a pod.

Na priloženom CD sa nachádzajú tieto súbory a adresáre:

- autorun.bat – dávkový súbor pre automatické spustenie inštalácie
- install.exe – inštalačný program aplikácie
- Diplomová práca – adresár obsahujúci túto diplomovú prácu vo formáte PDF
- Odkazy – odkazy na webové stránky použitých technológií

B Plné znenie zadania

V této základní verzi je editor schopen vytvořit uživatelské rozhraní podle popisu v XML nebo skriptu, je schopen provázat toto rozhraní s funkcemi napsanými ve skriptu, je možné v něm plně vytvářet a editovat prototypy a je možné z těchto prototypů vytvářet scény (levely hry). Editor v základní verzi obsahuje všechny skripty potřebné k vytvoření scény a dále je schopen vytvořit a editovat kolizní obálky.

K plnému pochopení prototypů, herních objektů a scén je třeba prostudovat dokumentaci k enginu hry.

Základní koncept

- Editor edituje prototypy a scény (levely hry). Scéna je složená z herních objektů, kde každý herní objekt je vytvořen podle nějakého prototypu.

- Podle předchozího bodu pracuje editor ve dvou základních módech: editace prototypu a editace scény.

- Editor lze dále přepnout do herního módu (tj. spustit hru) a to se zachováním již naeditovaného stavu.

- Prakticky každý objekt enginu (ve smyslu programátorském) má rozhraní, které lze volat ze skriptu a ví jak sám sebe uložit a načíst z disku (persistentní objekt). Objekty jsou dále uspořádány v hierarchii připomínající adresářovou strukturu (takovými objekty jsou prototyp, herní objekt, scéna atd.). Editace hry probíhá z větší části na úrovni manipulace s těmito objekty (nastavováním parametrů, voláním metod objektů a jejich přemísťováním v hierarchii) skrze skript. Pro uložení naeditovaného prototypu nebo scény se použije vlastnost persistence.

- Tam, kde by napsání skriptu, jenž provádí nějakou funkci editoru, nebylo možné nebo praktické nebo rychlostně únosné, tam je možné použít C++ kód. V tom případě je dobré naprogramovat nový skriptovatelný objekt s novou funkčností tak, aby jeho služby mohly později využívat jiné skripty.

- Uživatelské rozhraní je popsáno pomocí XML nebo skriptu (dále jen šablona). Je třeba zhodnotit výhodnost obou variant a jednu zvolit.

- Části (zejména specifické ovládací prvky), které by nebylo možné nebo vhodné takto popsat, je možné implementovat do C++ kódu.

- Popis uživatelského rozhraní a funkční skripty editoru, tj. jinými slovy sada šablon a sada skriptů tvoří jednu konfiguraci. Editor pak lze spustit s různými konfiguracemi (vždy jen s jednou). Konfigurace můžeme chápat (a budou tak také využívány) jako popis podoby a funkčnosti editoru pro daný typ hry (editor pak může být využíván pro různé typy her).

Prototypy

- Každý prototyp se skládá z jedné nebo více těchto částí: vizualizační geometrie, kolizní obálka, fyzikální těleso, herní charakteristiky a skript.

- Každá tato část obsahuje několik parametrů.

- V každé konfiguraci existuje vždy několik daných typů prototypů. Prototyp je popsán šablonou. Předpokládá se, že ke každému typu herního objektu (viz. dále) bude existovat jeden prototyp.

- Počet parametrů ke každému prototypu je vždy úplný (takže výsledný dialog pro editaci prototypu obsahuje vždy kompletní seznam parametrů, které u něj lze nastavit). Každý parametr má jednu nebo dvě defaultní hodnoty a to jednak defaultní hodnotu v editoru a defaultní hodnotu v enginu. Defaultní hodnota v editoru se zadává v šabloně prototypu a každý parametr jí má povinně. Defaultní hodnotu enginu mají jen některé parametry. Tato hodnota je vlastností C++ objektu a v šabloně se pouze uvádí, zda taková defaultní hodnota existuje a zda ji lze využít (neuvádí se už její konkrétní hodnota).

- Vizualizační geometrie je zadána jako odkaz na nSceneNode objekt (nebo nějaký potomek) uložený na disku.

- Bude možné označit prototyp jako tzv. multi-objekt. Zatímco u normálního objektu se vytváří z prototypu jen jeden herní objekt, v případě multiobjektu se z každého podřízeného nSceneNodu (ale jen na první úrovni) vytvoří jiný herní objekt. Například máme model města a chceme, aby každý dům byl uvažován jako samostatný objekt (kvůli řešení viditelnosti), potom označíme prototyp jako multi-objekt.

- Pravděpodobně bude třeba mít možnost provádět triviální transformace v model space daného objektu.

- Prototyp může sloužit jako kontejner pro jiné prototypy. Obsažené prototypy mohou být různého druhu a mohou být multiobject. Herní objekt se vytváří jen z prototypů obsažených v kontejneru, nikoliv z kontejneru samotného.

- Platí pravidlo, že multi-objekt prototyp nebo prototyp typu kontejner může být odkazován jen jednou. Editor toto pravidlo hlídá.

- Lze vložit jednu nebo více primitivních kolizních obálek do jednoho prototypu, lze je vizualizovat a lze na nich provádět triviální transformace.

- Lze vložit kolizní obálku typu polygon soup z externího souboru.

- Lze vytvořit kolizní obálku z vizualizační geometrie (prakticky se jen odstraní všechny složky vertexů kromě souřadnice) a uložit jí na disk. Prototyp pak obsahuje odkaz na soubor.

- Lze zadat hmotnost a typ rozložení hmoty u fyzikálního tělesa (fyzikální těleso je vlastně kolizní obálka s hmotností).

- Lze zadat herní charakteristiky (počet životů u postav, kadence u zbraní atd…).

- Lze připojit externí skript (např. skript k nášlapnému triggeru spustí animaci otevírání brány a odstraní kolizní těleso, jenž způsobovalo její neprůchodnost). Tvorba těchto skriptů je věcí editace hry, nikoliv součástí tohoto zadání.

Scény (levely)

- Do scény lze vložit libovolné množství herních objektů.

- Každý herní objekt má svůj předobraz v nějakém prototypu.

- Na každý herní objekt lze aplikovat triviální transformace, vyjma objektů, které vznikly z multi-objekt prototypů nebo z prototypů typu kontejner.

- Vnitřně přísluší každému prototypu jeden C++ objekt, takzvaný „Tvůrce“, který dokáže z jednoho prototypu vytvořit jeden nebo více herních objektů. Když se ukládá nebo načítá scéna (level), pak se ukládají nebo načítají právě tito Tvůrci. Před uložením levelu se z herních objektů vytvoří tvůrci a ti se uloží, po nečtení levelu s z tvůrců vytvoří herní objekty. Jinak řečeno, editují se sice herní objekty, ale ukládají a načítají se tvůrci. Více v dokumentaci k programovému kódu.

- Herní objekt lze zvolit klinutím na objekt ve scéně.

- Existuje seznam všech herních objektů ve scéně. V tomto seznamu lze vyhledávat (jako index help), lze filtrovat výpis objektů podle typu nebo prefixu, typy objektů jsou odlišeny buďto ikonou nebo barvou.

- Lze zvolit více objektů najednou, ať už kliknutím do scény nebo v seznamu (v seznamu se buďto zadá prefix a zvolí se všechny objekty s tímto prefixem a nebo se volí klikáním na jednotlivé objekty např. s klávesou Ctrl).

- Editor zajišťuje, že nebudou vytvářeny duplicitní jména objektů.

Herní objekty

- Předpokládá se, že ke každému hernímu objektu bude existovat jedna šablona a předpokládá se také, že bude existovat tolik šablon herních objektů, kolik bude šablon prototypů. Šablona k hernímu objektu zachycuje povolené změny vzhledem k prototypu. Obsahuje maximálně tolik parametrů a v takovém složení, jako prototyp. Parametry, které měnit nelze, v šabloně herního objektu obsažené nejsou.

- Na herní objekt lze aplikovat triviální transformace. Tyto transformace se aplikují na objekt jako celek (nelze je aplikovat například jen na kolizní obálku).

- Pokud je zvoleno více herních objektů, pak se zobrazí dialog jen s těmi parametry, které lze editovat u všech herních objektů ve výběru. Hodnoty parametrů se zobrazí jen tehdy, jsou-li pro všechny herní objekty ve výběru stejné. Každý parametr pak lze změnit u všech objektů ve výběru.

Uživatelské rozhraní

- Uživatelské rozhraní se skládá z hlavního menu (např. klasická lišta s roletovým menu nebo ikony..), seznamu herních objektů (editace scény) a dialogu s parametry prototypu (editace prototypu) nebo herního objektu (editace scény).

- Hlavní menu je popsáno v šabloně (položky včetně hierarchie pod položek).

- V šabloně je také uveden skript, který je volán při volbě položky menu. Tímto skriptem samozřejmě může být jiná skriptová funkce.

- Je třeba se zamyslet nad tím, zda je nezbytně nutné, aby bylo možné provázat položky menu s výkonnými částmi v zkompilovaném C++ kódu. Pokud bychom se tomu mohli vyhnout a spolehnout se jen na volání skriptů, bylo by to jistě elegantnější.

- Z hlavního menu je možné otevírat další dialogy (modální i nemodální), které jsou také popsané šablonou. Dialog popsaný šablonou obsahuje jen dané ovládací prvky (button, edit box, list…), tj. nevytváří nové. Takovýto ovládací prvek buďto nastavuje parametr nějakého objektu a nebo volá skript.

- Vzhledem k tomu, že dialog v případě editace prototypu nebo herního objektu slouží k nastavení parametrů C++ objektu. Je možné dialog provázat s C++ objektem existujícím v hierarchii objektů. K němu se pak naeditovaný parametr vztahuje.

- Některé speciální ovládací prvky nebo dialogy např. seznam herních objektů ve scéně nebo procházení disku budou vytvořeny v C++.

C Používateľská príručka

Systémové požiadavky

Aplikácia vyžaduje:

- Operačný systém: Windows 2000, XP alebo NT 4.0 a vyšší
- Pamäť: 512 Mb
- Procesor: Intel Pentium 4 1,5 GHz alebo ekvivalentný
- Grafická karta: 128 Mb s podporou pixel a vertex shaderov verzie 2.0
- CD-ROM

Inštalácia a spustenie aplikácie

Pri vložení priloženého CD do počítača sa automaticky spustí inštalačný program aplikácie. Ďalej je potrebné už len nasledovať kroky inštalačného programu. Po dokončení inštalácie je možné program spustiť pomocou odkazu v menu Štart systému Windows.

V prípade, že sa inštalácia nespustí automaticky, je potrebné spustiť súbor install.exe z CD manuálne.

Hlavné okno aplikácie

Hlavné okno editora obsahuje po spustení iba úvodnú ponuku. Pomocou príkazov z ponuky je možné otvoriť požadovaný prototyp, vytvoriť nový a pod. Mnohé príkazy majú tiež definované klávesové skratky. Tie sa štandardne zobrazujú vedľa názvu príkazu. Nasleduje zoznam všetkých príkazov s krátkym vysvetlením.

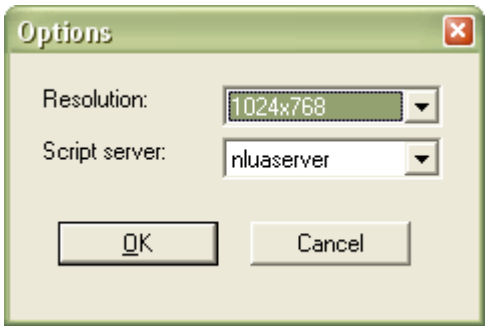
- File
  - New scene – vytvorí novú prázdnu scénu (level)
  - Open scene – otvorí existujúcu scénu z disku
  - New prototype – vytvorí nový prázdny prototyp herného objektu
  - Open prototype – otvorí existujúci prototyp z disku
  - Exit – ukončí aplikáciu
- Edit
  - Options – zobrazí dialóg s nastavením aplikácie
- Help
  - Contents – zobrazí súbor s používateľskou príručkou
  - About – zobrazí okno „O programe“

Nastavania aplikácie

Pomocou dialógu pre nastavenia aplikácie (príkaz Edit / Options) je možné nastaviť rozlíšenie (veľkosť hlavného okna) a názov triedy serverového objektu, ktorý bude použitý pri ukladaní prototypov a levelov. To ovplyvní výsledný skriptovací jazyk a formu súborov. Na načítavanie súborov toto nastavenie vplyv nemá.

Na výber je:

- nluaserver – súbory budú textové, zapísané pomocou jazyku Lua
- nbinscriptserver – súbory budú uložené v binárnej forme pomocou špeciálneho binárneho jazyka



Dialóg Options

Editácia prototypu

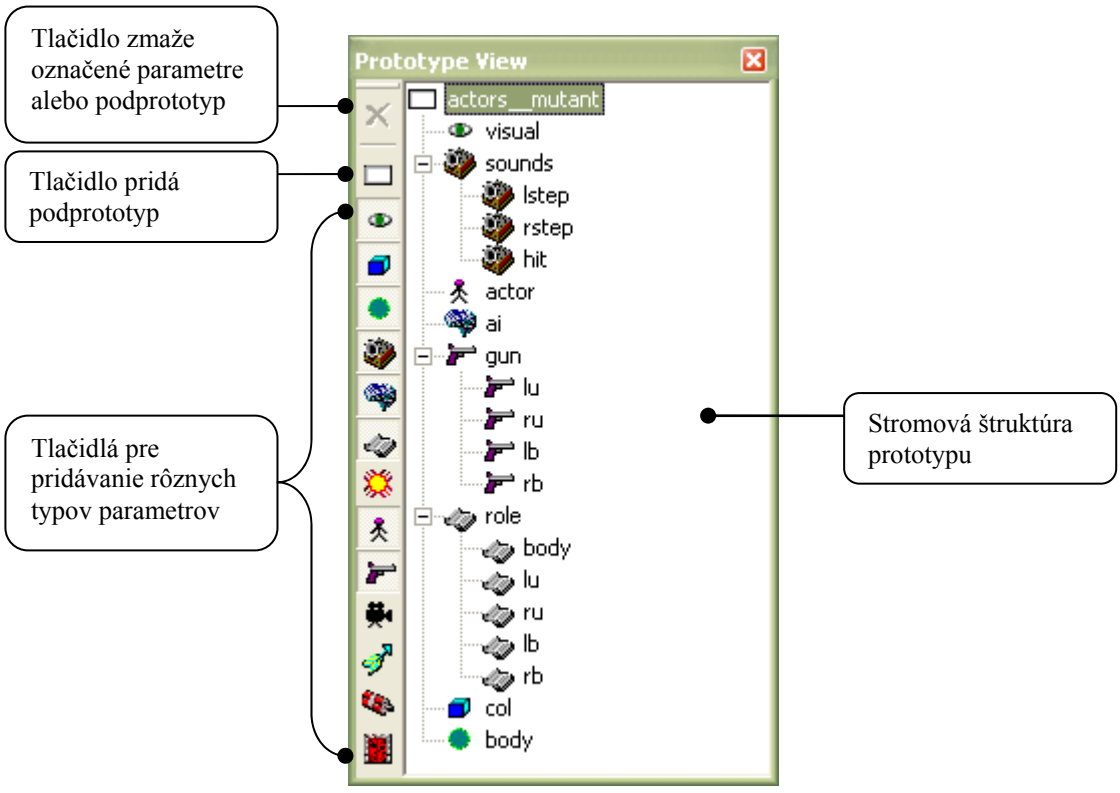
Pri vytvorení nového či otvorení existujúceho prototypu sa editor prepne do módu pre editáciu prototypu. Tým sa okrem iného zmení i hlavná ponuka. Nasleduje krátke vysvetlenie príkazov, ktoré ešte neboli zmieňované:

- File
  - Save – uloží otvorený súbor. Ak je súbor nový spustí sa príkaz Save As
  - Save As – uloží otvorený súbor pod novým názvom
  - Close – zavrie otvorený súbor
- Edit
  - Delete – zmaže označené parametre
- Transformation
  - Translation – nastaví posunutie ako aktívnu transformáciu
  - Rotation – nastaví rotáciu ako aktívnu transformáciu
  - Scale – nastaví zmenu mierky ako aktívnu transformáciu
- View
  - Ptototype View – zobrazí/skryje dialóg so stromovou štruktúrou prototypu
  - Properties – zobrazí/skryje dialóg s vlastnosťami označených parametrov
  - Developement Info – zobrazí/skryje monitorované systémové informácie
  - Bounded Boxes – zobrazí/skryje vizualizácie kubických obálok
  - Collision Envelopes – zobrazí/skryje vizualizácie kolíznych obálok
  - Occluders – zobrazí/skryje vizualizácie kvádrov pre optimalizáciu vykresľovania
  - Navigation Mesh – zobrazí/skryje vizualizácie navigačnej siete pre algoritmus vyhľadávania ciest
  - RPG Info – zobrazí/skryje RPG (Role Player Game) informácie objektov
  - Pick Objects – zobrazí/skryje vizualizácie objektov pre označovanie myšou (funkčné iba pri zostavení aplikácie pre ladenie)
  - Console – zobrazí/skryje systémovú konzolu príkazov
- Camera
  - Fly Mode – prepína mód kamery

- Speed Up – zvýši rýchlosť pohybu kamery
- Speed Down – zníži rýchlosť pohybu kamery

Pri editácii prototypu hrajú dôležitú rolu vstavané dialógy. Dialóg s názvom Prototype View zobrazuje stromovú štruktúru prototypu a parametrov. Jednotlivé typy parametrov sú od seba odlíšené ikonou. Dialóg obsahuje aj panel nástrojov s tlačidlom pre zmazanie označených parametrov a sadou tlačidiel pre pridávanie podprototypov a parametrov.

Platí, že prototyp môže obsahovať buď jednu úroveň podprototypov alebo parametre (nie však súčasne). Ďalej sa v prototype nemôže nachádzať dva a viac parametrov rovnakého typu. Parametre môžu rovnako obsahovať jednu úroveň podparametrov rovnakého typu. Všetky tieto pravidlá editor stráži. Napríklad pri pokuse pridať podprototyp do prototypu, ktorý už obsahuje parametre, dôjde najprv k vymazaniu všetkých parametrov.



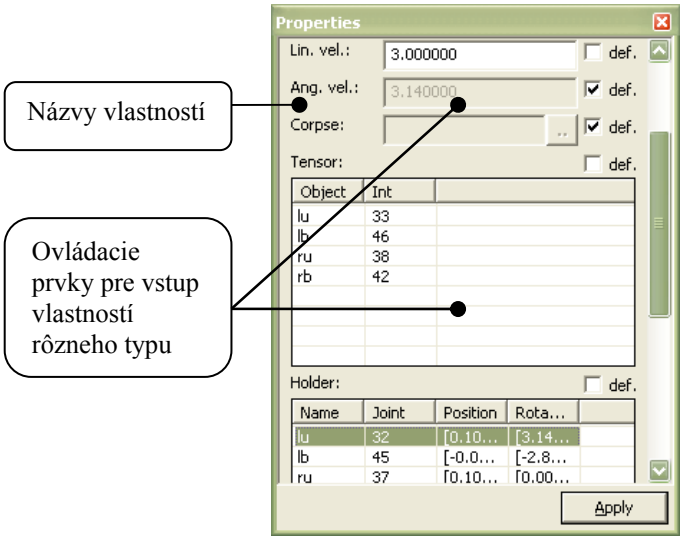
Dialóg Prototype View

Po pridaní parametrov či podprototypu ostane príslušné tlačidlo stlačené. Zmazať niektorú zložku prototypu je možné použitím tlačidla na paneli nástrojov alebo stlačením klávesu Delete na klávesnici.

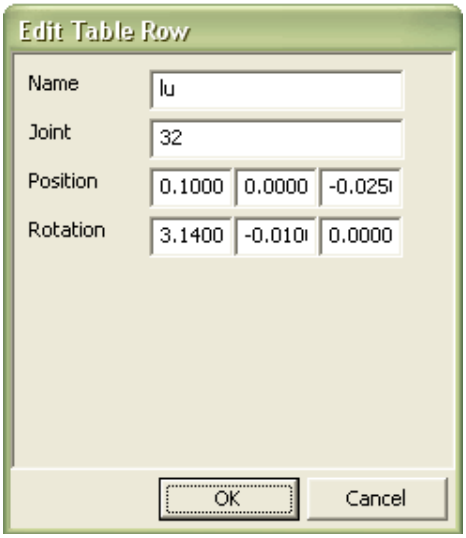
Vlastnosti označených parametrov sa zobrazujú v dialógu nazvanom Properties. Tu je možné vlastnosti parametrov zmeniť a aplikovať zmeny tlačidlom Apply. Pri aplikovaní nových vlastností sa kontroluje ich správnosť. V prípade, že je niektorá hodnota nesprávne zadaná, je na to používateľ upozornený. Žiadne zmeny sa neprejavia až pokým nie sú všetky hodnoty správne zadané.

Niektoré vlastnosti parametrov môžu mať aj formu tabuľky. Nové riadky sa do tabuľky pridávajú stlačením klávesu Insert, mažú stlačením Delete a existujúce riadky sa upravujú stlačením Enter alebo dvojitým klepnutím myšou na príslušný riadok. Tabuľka musí byť predtým samozrejme aktívna (aktivuje sa napr. klepnutím myšou kdekoľvek do tabuľky).

Vedľa políčok pre vkladanie hodnôt sa môže vyskytovať tlačidlo „def.“. Ak je zaškrtnuté, znamená to, že ako hodnota pre danú vlastnosť sa použije prednastavená hodnota, ktorá je súčasťou enginu. Ovládací prvok pre vloženie hodnoty je vtedy neprístupný. Ak nie je zaškrtnuté, je možné zadať vlastnú hodnotu. V prípade, že sa pri vlastnosti tlačidlo „def.“ nenachádza, hodnotu je nutné zadať ručne – je povinná.



Dialóg Properties.



Dialóg pre úpravu riadku tabuľky.

Väčšina dostupných parametrov nemá žiadnu viditeľnú zložku, ktorá by sa mohla zobraziť. Niektoré z týchto parametrov je možné vizualizovať pomocnými geometrickými útvarmi. Jedná sa napríklad o kolízne parametre. Pre zobrazenie pomocných útvarov stačí zvoliť z hlavnej ponuky príslušný typ. Kamerové a svetelné parametre používajú pre svoje zobrazenie systémové modely. Parametre ktoré je možné akýmkoľvek spôsobom zobraziť, je možné aj označiť pomocou myši priamo v zobrazovanej scéne.

Okolo zobrazovaného prototypu je možné sa pohybovať, teda meniť polohu kamery. Stlačením klávesu Space začne kamera „lietať“. Pohybom myši je možné meniť smer kamery, stlačenie ľavého alebo pravého tlačidla myši spôsobí pohyb kamery dopredu a dozadu. Rýchlosť pohybu kamery sa mení pomocou kláves + a -. Počas pohybu kamery je označovanie parametrov v scéne vypnuté. Opätovným stlačením Space kamera zastane.

Editácia scény

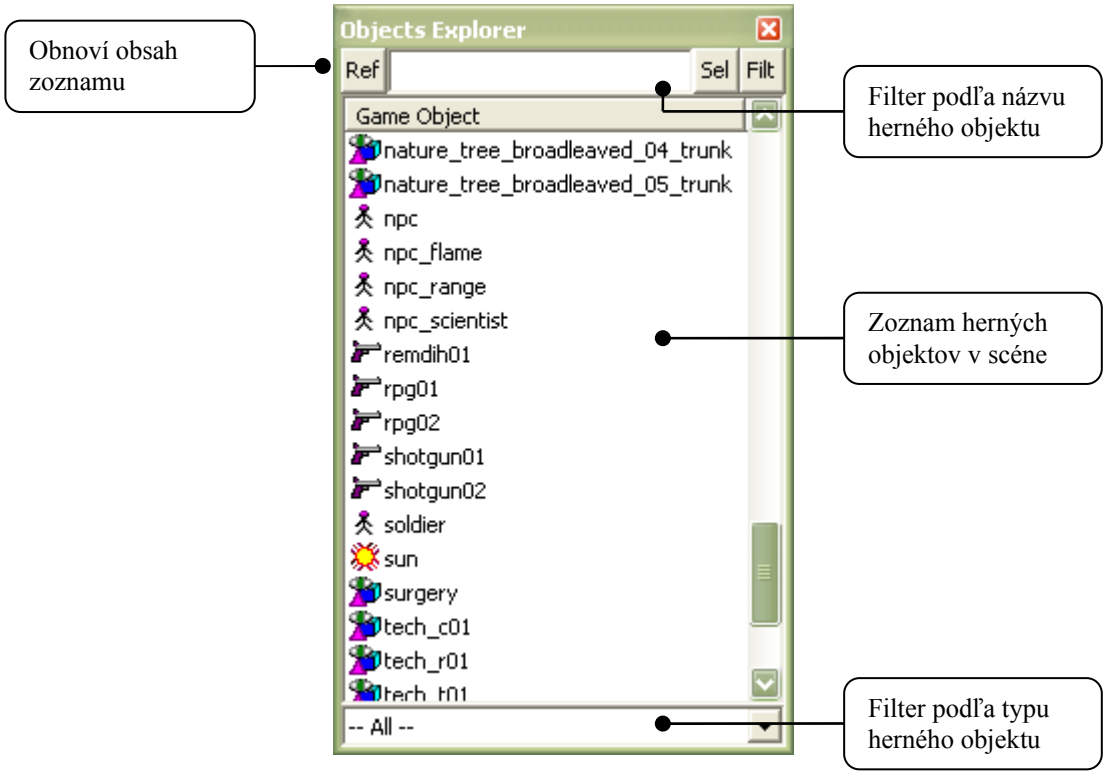
Otvorením levelu z disku alebo vytvorením nového levelu sa spustí mód editácie scény. Špecifickými príkazmi pre tento mód sú:

- File

- Open Object Prototype – otvorí prototyp označeného herného objektu pre editovanie. Ak je označených viac herných objektov, otvorí prototyp prvého z nich
- Play Mode – spustí herný mód pre otvorený level

- Edit
  - Add Object – pridá do scény nový herný objekt so zadaným názvom odvodený od zadaného prototypu
  - Delete – zmaže označené herné objekty
  - Setup Level – zobrazí dialóg pre nastavenie vlastností levelu
- View
  - Internal Light – zapne/vypne pomocné svetlo pre osvietenie objektov

Aj v tomto móde sú dôležité dialógy. Dialóg Properties je už známy. Opäť zobrazuje vlastnosti zvolených objektov. Herných objektov môže byť naraz označených viacero. V takomto prípade sa v dialógu Properties zobrazujú iba spoločné vlastnosti označených objektov.



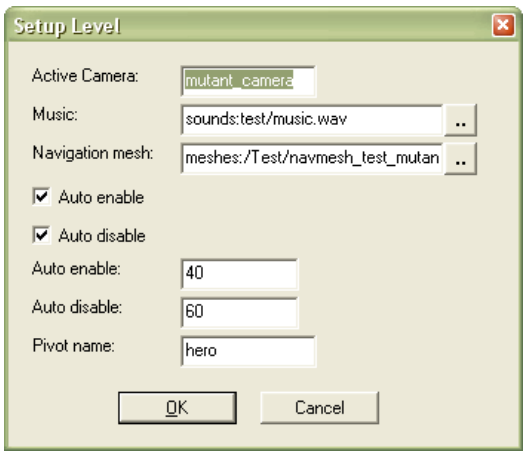
Dialóg Object Explorer

Novým dialógom je „Objects Explorer“. V tomto dialógu sa nachádza zoznam všetkých herných objektov v scéne, usporiadaný podľa názvu objektov. V scéne sa však môže nachádzať veľmi veľa objektov, preto dialóg ponúka možnosti pre ich filtrovanie. V spodnej časti sa nachádza filter podľa typu herného objektu. Zvolením konkrétneho typu z ponuky sa v dialógu zobrazia iba objekty príslušného typu. V hornej časti sa nachádza filter podľa názvu herného objektu. Do editačného poľa filtra sa zadáva vzorka pre názov objektov. Stlačením tlačidla Filt (Filter) budú v zozname zobrazené iba herné objekty, ktorých názov vyhovuje zadanej vzorke. Stlačenie Sel (Select) spôsobí označenie všetkých objektov, ktorých názov vyhovuje zadanej vzorke. Tlačidlo Ref (Refresh) aktualizuje zoznam (filter ostáva aktívny).

Obidva filtre môžu byť použité súčasne. Vzorka pre filtrovanie podľa názvu objektov môže obsahovať špeciálne zástupné znaky. Znak '?' znamená práve jeden ľubovoľný znak, znak '*' zastupuje ľubovoľný počet (i nulový) ľubovoľných znakov. Pri zadávaní vzorky záleží na veľkých a malých písmenách.

Samotný level obsahuje nastavenia, ktoré je možné zadať pomocou príkazu z ponuky Edit/Setup Level. V zobrazenom dialógu je možné nastaviť postupne:

- Aktívnu kameru – kameru, ktorá bude aktivovaná po načítaní levelu v hre.
- Hudbu na pozadí.
- Súbor s navigačnou sieťou pre algoritmus vyhľadávania ciest.
- Automatické zapínanie umelej inteligencie postáv.
- Automatické vypínanie umelej inteligencie postáv.
- Maximálna vzdialenosť pre automatické zapínanie umelej inteligencie postáv.
- Minimálna vzdialenosť pre automatické vypínanie umelej inteligencie postáv.
- Identifikátor postavy, ktorá predstavuje hlavného hrdinu.

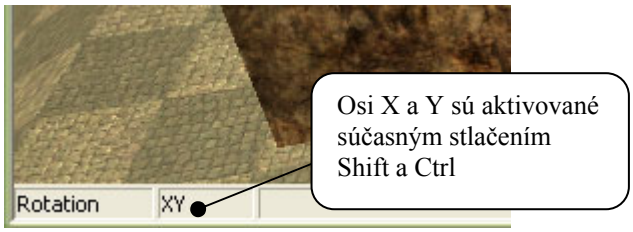


Dialóg Setup Level

V tomto móde sú podporované dve vizuálne transformácie herných objektov: posunutie a rotácia. Posunutie sa aktivuje klávesom T (translation), rotácia klávesom R (rotation). Zmena mierky (sacale) herného objektu momentálne podporovaná nie je, hoci sa už nachádza v ponuke. Aktuálne zvolená transformácia sa zobrazuje v ľavom dolnom rohu hlavného okna. Pre editáciu objektu je potrebné súčasne stlačiť kláves príslušný požadovanej osi a ľavé tlačidlo myši. Následný pohyb myši sa premieta do relatívnej zmeny transformácie. K jednotlivým osiam sú priradené tieto klávesy:

- Shift – os X
- Ctrl – os Y
- Alt – os Z

Tieto klávesy môžu byť stlačené aj súčasne v rôznych kombináciách. Aktivované osi sú opäť zobrazené v spodnej časti hlavného okna. Ľavé tlačidlo myši slúži pre aktiváciu vizuálneho editovania a pohyb myši pre zmenu hodnôt. Zmenené hodnoty sa okamžite zobrazujú v dialógu Properties.



Aktuálna transformácia a osi transformácie

Z módu pre editáciu scény je možné prejsť priamo do herného módu. Podmienkou je, že editovaný level musí obsahovať aspoň jednu kameru a musí byť uložený. V hernom móde je ovládanie editoru zamenené za ovládanie hry. Pomocou príkazu File/Close Play Mode je potom možný návrat do editácie scény. Poloha kamery a nastavenie aplikácie sa obnoví do stavu pri spustení herného módu.

Editor tiež podporuje otvorenie prototypu priamo z editácie scény. Stačí označiť objekt, ktorého prototyp chceme editovať a zvoliť príkaz File/Open Object Prototype. Po ukončení editácie prototypu sa aplikácia opäť vráti do pôvodnej scény, pričom zmeny v prototype sa aplikujú na príslušné herné objekty.

Konfigurácia editora

Konfigurácia editora predstavuje popis grafického používateľského rozhrania pomocou XML dokumentov spolu s funkčnými skriptami v jazyku Lua. Je možné definovať vlastné ponuky, dialógy s ovládacími prvkami, a šablóny pre zobrazovanie vlastností parametrov alebo herných objektov. Editor môže byť spustený s rôznymi konfiguráciami. Aktuálna konfigurácia je zadaná v súbore MutantGame\System\editor.n2. Tu sa nachádza cesta k hlavnému konfiguračnému súboru.

```
call('setconfiguration', [[editor:mutant.xml]])
```

Hlavný konfiguračný súbor

Tento súbor obsahuje zoznam ponúk, dialógov a šablón. Má nesledujúcu štruktúru:

```
<?xml version="1.0" encoding="WINDOWS-1250"?>
<editor>
     <menus>
          <menu src="Súbor ponuky1" />
          <menu src="Súbor ponuky2" />
          ...
     </menus>
     <dialogs>
          <dialog src="Súbor dialógu1" />
          ...
     </dialogs>
     <templates>
          <template src="Súbor šablóny1" />
          <template src="Súbor šablóny2" />
          <template src="Súbor šablóny3" />
          ...
     </templates>
```

```
</editor>
```

Texty „Súbor ponuky“, „Súbor dialógu“ a „Súbor šablóny“ predstavujú cesty k jednotlivým XML dokumentom.

Súbor ponuky

Základom XML súboru ponuky je značka <menu>.

```
<?xml version="1.0" encoding="WINDOWS-1250"?>
<menu id="Identifikátor">
      ...
</menu>
```

„Identifikátor“ je jednoznačný textový identifikátor ponuky. Vo vnútri značky <menu> môže byť ľubovoľný počet roletových ponúk <popup> alebo položiek <item>. Samozrejme, každá roletová ponuka môže obsahovať ďalšie ponuky, položky alebo separátory <separator />. Roletové ponuky i položky obsahujú povinne názov <caption>, položky môžu navyše obsahovať skript (alebo cestu k skriptovému súboru) <onclick>.

```
<popup>
      <caption>Názov</caption>
      <popup>
            ...
      </popup>
      <separator />
      <item>
            ...
      </item>
      ...
</popup>
<item>
      <caption>Názov</caption>
      <onclick>Skript</onclick>
</item>
```

Súbor dialógu

Hlavnou značkou súboru dialógu je <dialog>.

```
<?xml version="1.0" encoding="WINDOWS-1250"?>
<dialog id="Identifikátor" style="Štýl">
      <caption>Názov</caption>
      <position center="" absolute="" x="" y="" />
      <size width="" height="" />
      <controls>
      ...
      </controls>
</dialog>
```

„Identifikátor“ je jednoznačný textový identifikátor dialógu. „Štýl“ ma jednu z hodnôt:

- dialog – štandardný dialóg
- toolbox – dialóg so stenčeným záhlavím

Pozícia dialógu sa nastavuje atribútmi značky <position>. Konkrétne atribútmi x a y (celočíselné hodnoty) alebo atribútom center (pavdivostné hodnoty: true, false, yes, no, 0, 1) – poloha dialógu je automaticky centrovaná. Pozícia môže byť definovaná absolútne alebo relatívne vzhľadom k hlavnému oknu aplikácie (to platí i pre centrovanie).

Atribúty width a height značky <size> nastavujú veľkosť dialógu v pixeloch (celočíselné hodnoty).

Definície ovládacích prvkov sú uzatvorené značkou <controls>. Táto časť je spoločná pre dialógy aj šablóny. Presný popis sa nachádza nižšie.

Súbor šablóny

Šablóny uchovávajú ovládacie prvky pre nastavovanie vlastností parametrov alebo herných objektov. Identifikátor šablóny musí byť rovnaký ako meno triedy, ku ktorej šablóna patrí.

```
<?xml version="1.0" encoding="WINDOWS-1250"?>
<template id="Meno triedy">
      <controls>
      ...
      </controls>
</template>
```

Hlavná značka <template> obsahuje iba sekciu ovládacích prvkov <controls>.

```
<controls>
      <control id="Identifikátor" type="Typ" target="Vlastnosť"
                defvalue="">
            <caption>Caption</caption>
            <position x="" y="" />
            <size width="" height="" lblindent="" />
            ...
      </control>
      ...
</controls>
```

Každý ovládací prvok <control> má svoj textový identifikátor a typ. Atribút target predstavuje cieľovú vlastnosť parametrov či herného objektu. V prípade, že target nebude odpovedať žiadnej vlastnosti, bude ovládací prvok neprístupný. Atribút defvalue (pravdivostná hodnota) vyjadruje, či je možné využiť prednastavenú hodnotu z enginu. Posledné dva menované atribúty sa nepoužívajú v prípade dialógov.

Tiež je možné definovať názov ovládacieho prvku, pozíciu a veľkosť. Značka <size> obsahuje jeden špeciálny atribút: lblindent (label indent). Atribút špecifikuje veľkosť priestoru medzi zobrazeným názvom a samotným ovládacím prvkom. Šírka a výška tu znamenajú celkovú veľkosť ovládacieho prvku spolu s názvom a ďalšími súčasťami.

Ovládacie prvky pri použití v šablónach môžu mať definovanú sadu skriptov:

```
<control ...>
      <onrefresh>Aktualizačný skript</onrefresh>
      <onapply>Aplikačný skript</onapply>

      <ontranslate>Skript posúvania</ontranslate>
      <onrotate>Skript rotácie</onrotate>
      <onscale>Skript mierky</onscale>
      ...
</control>
```

Aktualizačný skript je volaný editorom pri potrebe preniesť hodnotu z parametrov/objektu do ovládacieho prvku. Aplikačný skript sa používa pre opačný smer – z ovládacieho prvku do parametrov/objektu.

Ďalšie skripty sú využité pri vizuálnej editácii. Tieto skripy musia obsahovať dva špeciálne symboly: %s a %f. Symbol %s bude pred zavolaním skriptu nahradený za definíciu osi: axisx, axisy alebo axisz; symbol %f bude nahradený za relatívnu zmenu hodnoty príslušnej transformácie.

Ostatné nastavenia ovládacích prvkov závisia na type prvku. Editor podporuje tieto typy ovládacích prvkov:

- label
- variant
- button
- input-bool
- input-string
- input-int
- input-float
- input-vector3
- input-list
- input-file
- table

label

Jednoduchý statický text bez hodnoty. Tento ovládací prvok nemá žiadne ďalšie nastavenia.

```
<control type="label" ...>
      ...
</control>
```

variant

Text, ktorý je schopný zobraziť hodnotu akéhokoľvek typu (textový reťazec, číslo, pravdivostná hodnota, trojzložkový vektor). Prednastavenou hodnotou definovanou v značke <value> môže byť iba text.

```
<control type="variant" ...>
      <value text="Prednastavená hodnota" />
      ...
```

```
</control>
```

button

Jednoduché tlačidlo. Pomocou značky <onclick> môže mať definovaný skript pre udalosť stlačenia. Atribút modalresult nastavuje hodnotu, ktorú vráti dialóg zobrazený modálne, ak bolo stlačené toto tlačidlo. Dialóg sa okrem toho automaticky zavrie. Hodnotou atribútu modalresult môže byť: mrok, mrcancel, mryes, mrno, mrabort, mrretry a mrignore.

Tiež je možné označiť tlačidlo ako prednastavené pre rodičovský dialóg. To znamená, že stlačenie klávesu Enter kdekoľvek v dialógu spôsobí stlačenie tohto tlačidla. Túto vlastnosť má na starosti atribút default (pravdivostná hodnota). Podobne funguje atribút cancel. Rozdiel je iba v tom, že tlačidlo reaguje na stlačenie klávesy Esc.

```
<control type="button" default="" cancel="" modalresult="" ...>
     <onclick>Skript stlačenia</onclick>
     ...
</control>
```

input-bool

Zaškrtávacie políčko používané pre pravdivostné hodnoty.

```
<control type="input-bool" ...>
     <value value="" />
     ...
</control>
```

input-string

Editačné pole akceptujúce textové reťazce.

```
<control type="input-string" ...>
     <value text="" />
     ...
</control>
```

input-int

Editačné pole akceptujúce celočíselné hodnoty.

```
<control type="input-int" ...>
     <value value="" />
     ...
</control>
```

input-float

Editačné pole akceptujúce reálne čísla.

```
<control type="input-float" ...>
```

```
      <value value="" />
      ...
</control>
```

input-vector3

Tento ovládací prvok pozostáva z troch editačných polí. Používa sa pre nastavenie trojzložkových vektorov.

```
<control type="input-vector3" ...>
      <value value="" />
      ...
</control>
```

input-list

Ovládací prvok s vysúvateľnou ponukou textových položiek. Položky sú zadávané značkou <item />. Prednastavená hodnota by mala byť jedna z položiek.

```
<control type="input-list" ...>
      <value text="Položka 2" />
      <item text="Položka 1" />
      <item text="Položka 2" />
      <item text="Položka 3" />
      ...
</control>
```

input-file

Editačné pole kombinované s tlačidlom pre prechádzanie súborov na disku. Ak používateľ klepne na toto tlačidlo, zobrazí sa dialóg pre otvorenie súboru. Pre tento dialóg je možné, pomocou značky <initdir>, zadať úvodný adresár a pomocou značky <filter> povolené prípony súborov.

```
<control type="input-file" ...>
      <value text="" />
      <initdir>Úvodný adresár</initdir>
      <filter>Filter</filter>
      ...
</control>
```

Filter má nasledujúcu formu:

<Meno filtra1>|<Vzorka1>|[<Meno filtra2>|<Vzorka2>|[...]]|

Napríklad:

```
<filter>Mutant Model (*.n2)|*.n2||</filter>
```

Dva znaky '||' na konci filtra sú dôležité a netreba ich zabudnúť.

table

Najkomplikovanejším ovládacím prvkom je tabuľka. Zobrazuje používateľsky definované stĺpce pričom každý stĺpec môže mať ľubovoľný typ hodnoty okrem tabuľky. Stĺpce tabuľky sa definujú v podobe ovládacích prvkov. A to nie len preto, že stĺpce používajú rovnaké nastavenia, ale príslušné ovládacie prvky sa reálne vytvárajú v pomocnou dialógu pri editovaní riadku tabuľky.

```
<control type="input-table" ...>
      <controls>
            <control type="">
                  <caption>Názov stĺpca1</caption>
                  ...
            </control>
            <control type="">
                  <caption>Názov stĺpca2</caption>
                  ...
            </control>
            ...
      </controls>
      ...
</control>
```

Systémové objekty

Editor umožňuje registrovať nové typy (triedy) herných objektov a parametrov bez nutnosti zásahu do zdrojového kódu. Tiež je možné definovať systémové modely pre vizualizáciu parametrov/objektov.

Herné objekty

Jediná vec, ktorá je potrebná, je registrovanie novej triedy a jej ikony v dialógu Objects Explorer. Najlepšie miesto pre túto registráciu je v skripte MutantGame\Scripts\System\setup_editor.lua. Stačí pridať príkaz

g_ptr_objectsexplorer:addobjectclass("
<meno triedy>",
"<súbor ikony>",
"<súbor ikony pre uzamknutý objekt>")

na miesto po vytvorení dialógu. Napríklad:

```
g_ptr_objectsexplorer:addobjectclass(
    "mutgoactor",
    "editor:icons/actor.ico",
    "editor:icons/actor_locked.ico")
```

Parametre

Tu je situácia veľmi podobná ako pri herných objektoch. Nová trieda sa registruje pomocou dialógu Prototype View príkazom:

g_ptr_prototypeview:addparamclass(
"<meno triedy>",

"<súbor ikony>")

po vytvorení dialógu.

```
g_ptr_prototypeview:addparamclass(
    "mutprmactor",
    "editor:icons/actor.ico")
```

Systémové modely

Systémové modely pre všetky triedy spravuje samotný editor. Nový systémový model sa pridáva príkazom:

nebula.game:addsystemmodel("<meno triedy>", "<súbor modelu>")

v skripte setup_editor.lua.

```
nebula.game:addsystemmodel(
    "mutprmcamera",
    "models:system/camera.n2")
```